FORT : 파일 시스템 강인성 테스트 프레임 워크
(Framework of File System Robustness Test)

김 영 진 † 원 유 집 †† 김 락 기 †
(Young-Jin Kim) (You-Jip Won) (Ra-Kie Kim)

이 모 원 ††† 박 재 석 ††† 이 주 훤 †††
(Mo-Won Lee) (Jae-Seok Park) (Joo-Wheun Lee)

요 약 현대 저장장치는 점차 소형화, 대용량화 되어감에 따라 디스크의 집적도는 크게 증가 되어지고 있어 물리적인 오류가 발생하면 많은 디지털 정보가 소실 될 수 있다. 따라서 이에 대한 대처의 필요성이 점차 증가되어 본 논문에서는 파일시스템이 물리적인 오류에 대한 강인성과 무결성을 보장하고 있는지를 검증하는 테스트 프레임 워크를 제안하였다. 이와 같은 테스트프레임워크를 위해 디스크 상에 실제 베드섹터를 만들 수 있는 도구 및 저장장치에서 정의하고 있는 물리적인 오류를 모두 발생시킬 수 있는 도구를 각각 개발하였고, 파일시스템에서 저장장치에 읽기/쓰기 동작이 수행되는 상황을 즉시 모니터링 할 수 있는 도구를 추가 개발하여 FORT라는 이름의 파일시스템 강인성을 검증하는 테스트 프레임 워크를 최종적으로 구성하였다. FORT를 이용해서 신뢰성 및 견고성을 위주로 개발된 EXT3파일시스템을 대상으로 강인성의 수준을 나타낼 수 있는 분석표를 작성하였으며, 이를 분석하였다. 또한 파일시스템과 디바이스 드라이버 영역의 계층화된 구조를 융합할 수 있는 도구를 개발하여 획기적인 지능형 시스템으로 발전할 수 있는 방향을 제시 하였다.

키워드 : 파일 시스템, 강인성, 테스트 프레임 워크, 저장장치, 물리적인 오류, 베드섹터

Abstract Capacity of modern storage devices is becoming larger than yesterday and integration of disk is increasing. It refers that physical errors can damage a lot of digital information on storage devices. So we propose file system test framework in this paper to test integrity and robustness of file systems. We develop the tool for generating bad sectors on disks and the tool which creates all physical errors defined in storage devices. We also develop the tool for immediately monitoring the condition of read and write execution on storage devices. So, by integrating those tools, we develop FORT, test framework for confirming robustness of file system. We analyze robustness of ext3 file systems by FORT. Lastly, we present draft of intelligent system merging file system and device driver's layer architecture.

Key words : File system, Robustness, Integration, Storage, Physical error, Badsector

1. 서 론

1.1 연구동기

저장장치의 개발 분야는 대용량화, 디스크의 소형화, 고속화 위주로 연구 개발 되고 있으며, 저장장치에 적합하고, 효율적인 파일시스템 개발 또한 함께 진행되고 있다. 하지만 디스크 기반의 파일시스템에 대한 강인성 및 무결성을 검증하는 연구[1]가 진행 되고는 있으나 아직 많이 부족한 상태이다. 기존에는 저장장치에 맞는 파일시스템이 개발 되었을 경우, 무결성과 강인성을 검증하기 위해서 몇 가지 벤치마킹 도구[2]를 사용하고 있다.

· 이 논문은 2007년도 두뇌한국21 사업에 의하여 지원되었음
† 학생회원 : 한양대학교 전자통신컴퓨터공학과 연구원
yjkim@ece.hanyang.ac.kr
daybreak24@ece.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자통신컴퓨터공학과 교수
youjip.won@gmail.com
††† 비 회 원 : 한양대학교 전자통신컴퓨터공학과 연구원
a287848@ece.hanyang.ac.kr
elecwing@ece.hanyang.ac.kr
wheun@ece.hanyang.ac.kr
논문접수 : 2006년 10월 24일
심사완료 : 2007년 5월 23일

가장 대표적이고 널리 사용되는 것으로는 리눅스와 주변장치 간의 동작에 이상이 없는지 확인 할 수 있는 LTP[3](Linux Test Project)를 들 수 있다. LTP와 같은 벤치마크 도구는 파일시스템의 논리적인 오류 및 강인성을 검증하고, 파일시스템과 저장 장치간의 효율성에 대한 검증 또한 그 목적으로 하고 있다. 하지만 현대의 디스크는 빠르게 소형화 되고 대용량화 되어감에 따라 일정공간을 기준으로 보았을 때 데이타의 집적도는 높아지고 있다. 따라서 저장장치상에 발생하는 물리적인 오류를 그대로 방치할 경우 많은 디지털 정보가 소실될 수 있기 때문에 LTP에서와 같은 논리적인 오류 검사뿐만 아니라 물리적인 오류 검사에 이르기까지 검증 범위를 확대할 필요성이 날로 증가하고 있다. 논리적인 오류에 대한 파일시스템 프레임워크는 기존에 많은 연구[2]가 진행되어 왔으나, 저장장치와 얽혀있는 물리적인 오류[4]에 대한 연구는 아직 미흡한 상태이다. 왜냐하면 상위 계층인 파일시스템과 하위 계층인 저장장치가 계층적인 구조이며 각각의 계층별 연구가 독립적으로 수행되고 있기 때문이다. 그러나 저장장치는 독립적으로 디지털 정보를 읽고 쓸 수 없으며, 파일시스템 또한 디지털 정보를 독립적으로 읽고 쓸 수 없는 서로 밀접한 관련성을 가지고 있기 때문에 하나의 융합된 계층으로 생각해야 한다. 특히 물리적인 오류는 특정 계층에만 영향을 미치는 요소가 아니기 때문에 융합된 계층으로 다루어질 필요가 명확히 존재한다. 그래서 시스템 개발에서 검증 과정을 하나의 계층으로 보고 진행해야 마땅하다. 하지만 대부분의 시스템 개발자나 검증자는 파일시스템과 저장장치를 독립적인 영역으로 보고 있고 파일시스템 오류에 대한 연구에서도 물리적인 오류에 대한 검증은 포함되고 있지 않다.

따라서 본 논문에서는 파일시스템과 저장장치의 물리적인 오류에 대한 대처수준을 검증하기 위해 새로운 개념의 도구를 개발하였으며 이를 이용한 파일시스템 테스트 프레임워크를 형성하였다. 또한 개발된 파일 시스템 테스트 프레임워크를 실제 파일시스템에 적용하여 파일시스템의 물리적인 오류에 대한 강인성 및 무결성을 검증해보았다. 더불어 계층화된 파일시스템과 저장장치 사이를 효율적으로 융합할 수 있는 새로운 개념의 모듈을 개발하였고 물리적인 오류에 대해 보다 강인한 시스템을 형성할 수 있도록 제안하였다.

1.2 관련연구

SGI, Wipro, Technologies와의 합동 프로젝트로써 리눅스의 신뢰성, 강도, 안정성을 테스트 하는 오픈 소스 커뮤니티에 테스트 항목을 제공하는 LTP[3]가 현존하는 가장 대표적인 검증 벤치마킹 도구 이다. LTP에는 파일시스템 및 메모리 관리, 스케줄러 등의 부하측정 테스트로 구성되어 있으며, 커널 테스트의 자동화를 실현시켜 리눅스 커널의 성능향상을 목적으로 하고 있다. 하지만 LTP의 테스트 전략은 초기테스트와 부하측정 테스트전략만을 제공하고 있다. 초기 테스트는 하드웨어와 운영체제 간에 24시간 동안 테스트 항목을 만족해야 하고, 부하측정 테스트인 경우 시스템 사용도를 높게 하여 제품의 강도를 검증하는 기법이다. 하지만 커널의 논리적인 오류와 커널의 견고성을 검증할 뿐 저장장치의 물리적인 오류에 대해서는 테스트 항목조차 마련되어 있지 않다.

일반적으로 LTP의 부하측정 테스트 중에서 파일시스템 부하측정 테스트가 파일시스템의 견고성 및 강인성을 검증하기 위해 많이 사용된다. 이와 같은 테스트는 본질적으로 저장장치와 파일시스템 간의 유기적으로 얽혀있는 물리적인 오류[4]에 대한 테스트는 할 수 없다. 그래서 저장장치의 물리적인 오류에 대한 연구 필요성이 부각 되면서 물리적인 오류에 대한 분석, 대처 방안 및 저장장치의 물리적인 오류에 대한 파일시스템의 강인성 테스트 연구가 활발히 진행되고 있다. 이와 같은 연구의 핵심은 물리적인 오류 발생 기법이다. 디스크 상에 실제 물리적인 오류를 발생시킨 상태에서 검증이 이루어져야 가장 현실적이고 정확한 방법이 되겠지만 기존에 연구가 진행되고 있는 방향은 파일 시스템 영역에서 디바이스 드라이버 영역으로 보내지는 요청 큐 자료구조를 수정하는 등의 디바이스 드라이버 영역에서 인위적으로 물리적인 오류를 발생한 것과 같은 효과를 주는 것이 보편적인 기존 방식이다. 하지만 더 많은 연구가 점차 진행된다면 이와 같은 방식의 한계를 느끼게 될 것이며 보다 현실성, 정밀성, 정확성을 고려한 검증 방식을 찾게 될 것이다. 물론 디바이스 드라이버 영역에서 인위적인 조작으로 발생한 오류 상황이 실제 물리적인 오류의 발생 상황과 다르지는 않다. 다만 인위적인 조작에 의한 것이 아닌 최대한 현실에 가깝고 정밀한 검증을 위해서, 그리고 저장장치의 집적도가 높아지는 상황 속에서 반드시 현실 상황과 동일한 환경에서 파일시스템의 물리적인 오류에 대한 강인성 검증이 필요하게 될 것이다.

이 밖에 물리적인 오류에 대한 파일시스템의 대처 수준을 판단하는 연구뿐만 아니라 물리적인 오류 및 논리적인 오류를 발견할 수 있는 각종 유틸리티 개발 또한 연구 되고 있다. 가장 대표적인 유틸리티로는 FSCK[5](File System Check)를 들 수 있다. FSCK는 본질적으로 물리적인 오류 검사나 예방책이 아닌 논리적 / 물리적인 오류로 인해 파일시스템의 깨진 무결성을 복구 시켜주는 사후처리 역할을 한다. 하지만 FSCK를 수행하기 위해서는 파일시스템이 마운트 되어있지 않아야 하며, 그 수행시간이 짧지 않다는 단점을 가지고 있다. 이와 같이 사후 처리를 위한 유틸리티가 있는 반면 파일

시스템을 형성하기 전에 사용하는 MKFS[6]와 같이 오류를 예방하기 위한 유틸리티도 있다. MKFS는 파일시스템을 형성하기 위해서 저장장치 포맷 옵션을 통해 저장장치의 무결성을 검증할 수 있도록 개발되었다. MKFS에서 사용하고 있는 저장장치의 무결성 검증 방법으로는 파일시스템 포맷 전에 현재의 디스크의 내용을 전부 읽기 수행하여 정상적인 동작을 하고 있는지 확인하는 방법과 저장장치상에 임의의 데이타를 전부 써봐서 정상적인 쓰기 수행이 되는지 확인하는 방법이 있다. 마지막으로 이 두 가지 기법을 동시에 사용하는 방법으로써 임의의 패턴을 저장장치에 쓴 후 다시 읽어 정상적으로 읽혀졌는지 확인하는 기법을 사용하고 있다. 하지만 이러한 유틸리티의 연구 개발은 물리적인 오류에 대한 발생을 허용하는 것이며, 파일시스템이 형성될 때 저장장치를 검사하는 수준일 뿐 실제 파일시스템이 정상적으로 작동될 때의 무결성 검증은 하지 못하고 있다.

따라서 데이타 무결성에 위험을 줄 수 있는 원인 및 대처 방법에 대한 폭 넓은 연구[2]가 진행되고 있다. 무결성을 깨트리는 원인으로는 물리적인 오류 발생과 같은 하드웨어 오류와 논리적인 오류 발생과 같은 소프트웨어 오류 발생으로 크게 구분하고 있으며, 그 밖에 부주의한 사용자의 실수 및 악의가 있는 활동에 대한 오류 발생을 그 범위로 규정할 수 있다. 이러한 오류를 대처 할 수 있는 Tripwise[7,8], fsck[5]와 같은 유틸리티와 그 밖에 디스크의 미러링, 페리티 검사, 체크섬[2] 등도 디스크의 오류 발생에 대한 대처 방안으로 제시되고 있다. 하지만 이러한 유틸리티 및 대처 방안들은 저장장치의 오류가 이미 발생한 후에 이루어지거나 물리적인 오류 발생에 대한 피해를 최소로 하기 위한 회피책에 불과하다. 근본적으로 물리적인 오류에 대한 강인성을 검증할 수 있는 유틸리티 및 제안된 사항은 많지 않다[9].

본 논문에서는 저장장치에서 발생 할 수 있는 물리적인 오류에 대한 파일시스템의 올바른 대처 수준을 확인할 수 있도록 실제 저장장치 상에 물리적인 오류를 발생시킬 수 있는 도구와 디바이스 드라이버를 이용한 가상의 물리적인 오류 발생도구를 개발 하였다. 그 밖에 파일시스템의 물리적인 오류를 테스트 할 수 있는 도구를 개발하여 테스트 프레임 워크를 최종적으로 제안하고, 이와 동시에 파일시스템의 강인성을 향상시킬 수 있는 개발 도구를 간략히 소개하였다.

2. 파일시스템과 저장장치 오류

2.1 저장장치 오류의 발생원인과 종류

저장장치에서 발생 할 수 있는 물리적인 오류[4]를 파일시스템에서 어떻게 대처하는지 알기 위해서는 우선 오류를 발생시키는 원인부터 파악해야 한다. 첫 번째 원인은 가장 흔히 발생하고, 경미한 경우로써 방치할 경우 한 순간에 모든 데이타가 손상될 수 있는 베드섹터이다. 파일이 위치한 영역에 베드섹터가 발생할 경우 파일의 접근 및 실행이 안되며, 컴퓨터가 자주 다운되어 정상적인 작업이 수행되지 않는 오류이다. 두 번째 원인은 디스크의 충돌과 긁힘이다. 하드디스크가 아주 빠른 속도로 작업을 진행하던 도중 충격을 받거나 디스크의 헤드[10]와 디스크 사이의 간격에 끼어 드는 오염 물질에 의해 발생되는 것으로써 데이타의 복구가 가장 어려운 오류이다. 세 번째 원인은 하드디스크를 구성하는 스핀들 모터[10] 자체의 손상이다. 이 경우 하드디스크를 전부 분해해서 다시 조립해야 하는 복잡한 복구 과정이 필요하다. 네 번째 원인은 사용자의 실수로 하드 디스크를 떨어트리는 경우로써 하드디스크의 스핀들 모터 축이 변형되고, 진동 및 디스크 내부의 불균형이 일어나서 하드디스크의 준비상태가 되지 않는 경우이다. 다섯 번째 원인은 시스템의 전원장치에서 전달되는 전기적인 문제로 디스크 전체에 발생되는 오류가 있으며, 여섯 번째 원인은 디스크와 호스트 컴퓨터 사이에 물리적인 접촉 장치의 문제로 데이타 전달이 지연되거나 손실이 발생하는 경우이다. 이와 같이 몇 가지의 대표적인 물리적인 오류뿐만 아니라 많은 예기치 않은 원인으로 저장장치

표 1 파일시스템에서 수행하는 오류에 대한 발견 및 복구 수준정의

구분	수준 정의	정의된 수준 설명
오류 발견 수준	무시	저장장치에서 오류가 발생하더라도 파일시스템은 알지 못함
	에러코드	디바이스 드라이버 영역에서 파일 시스템으로 에러 코드를 전달
	특정 자료구조 검사	파일 시스템의 슈퍼블록 검사와 같은 방식으로 특정 자료구조 검사
	검사합[2]	에러 검출 코드를 이용한 발견
오류 복구 수준	무시	발생된 오류에 대한 어떠한 조치도 하지 않음
	에러 전달	사용자에게 오류 상황을 전달
	정지	파일시스템이 정지하거나 Read-Only로 전환
	재시도	해당 동작을 재시도
	논리적인 검사	Fsck[5]와 같이 논리적으로 파일시스템을 복구
	복제 기법[2]	특정 영역을 미리 복제하고, 오류가 발생했을 경우 복제본으로 복구

의 무결성은 깨질 수 있다.

2.2 저장장치의 오류에 대한 파일시스템의 대처

저장장치의 물리적인 오류는 많은 원인에 의해 발생할 수 있으며, 시스템의 계층적인 구조로 인해 물리적인 오류 정보가 파일시스템으로 전달되지 않아 파일시스템의 직접적인 오류로 발전될 수 있음을 예상하였다. 따라서 본 논문에서는 저장장치상에 오류가 발생했을 때, 파일시스템에서의 해당 오류에 대한 처리를 발견수준과 복구 수준으로 구분하여 정의 하였다.

저장장치의 물리적인 오류가 발생했을 경우 파일시스템 영역에서 해당 오류를 발견[11,12]하기 위한 파일시스템의 행동을 4가지로 정의하였다. 오류에 대해 전혀 인지하지 못하고, 파일시스템의 운용을 그대로 진행 할 경우 파일시스템은 물리적인 오류에 대해 어떠한 대처도 하지 않으므로 물리적인 오류에 대해 강인성이 약하다고 표현할 수 있으며, 디바이스 드라이버 영역[13]에서 에러코드를 전달 받아 합당한 처리를 진행하거나 검사합[2] 또는 특정 자료구조의 검사를 통해 오류를 탐지할 경우에는 물리적인 오류에 대한 강인성이 높다고 표현할 수 있다.

오류 발견에 이어 복구 과정을 그 수준에 따라 6가지로 정의하였다. 물리적인 오류가 발견되었음에도 불구하고 어떠한 복구 과정을 수행하지 않는 무시 복구 수준의 경우는 물리적인 오류에 대한 복구 측면에서 매우 약하다고 표현할 수 있으며, 미리 일정 영역을 복제하고 오류가 발생했을 때 복제본을 통해 복구하는 경우를 복구 측면에서 매우 강하다고 표현할 수 있다. 이와 같이 저장장치 오류에 대한 발견과 복구 수준을 세분화하여 분류 하였다.

테스트 프레임 워크를 파일시스템에 적용하여 저장장치 오류에 대한 발견과 복구수준을 각각 표현할 때 위와 같이 분류한 항목을 기준으로 파일시스템의 행동수준을 지칭할 것이며, 이를 통해 해당 파일시스템의 저장장치 오류에 대한 강인성 수준을 나타낼 것이다. 이와 같은 물리적인 오류에 대한 발견 및 복구 수준표는 기존에 많은 연구에서 사용되어지고 있으며 본 논문에서도 반영하고자 한다[9].

3. 지능형 오류 검출 방법

지금까지 저장장치에 발생되는 물리적인 오류에 대한 종류와 그 심각성을 살펴보았으며 파일 시스템 상에서 물리적인 오류에 대해 대처할 수 있는 수준을 나누어 보았다. 이와 같이 물리적인 오류의 발견과 복구 수준을 나누고 파일시스템을 검증 하는 이유는 분명 물리적인 오류에 대한 시스템의 문제점이 있기 때문이다. 따라서 본 논문에서 제시하는 파일시스템 강인성 테스트 프레임 워크를 이용해서 파일시스템을 검증해본다면 해당 파일시스템이 물리적인 오류에 대해 얼마나 취약한지 결과표를 통해서 명확히 알 수 있을 것이다.

파일시스템에서 물리적인 오류를 발견조차 하지 못하는 가장 큰 이유를 연구해본 결과 앞에서 언급했듯이 시스템 구조가 계층화되어 있기 때문이다. 물론 물리적인 오류가 하위영역에서 발견이 되고, 상위 영역으로 전달되어 합당한 오류 처리 작업이 진행된다면 이상적이겠지만 철저한 계층구조 속에서 정보 전달이 이루어지고 있지 않은 점이 문제였다. 이와 같은 시스템 구조를 바꿀 수 있는 또 다른 시스템이나 모듈이 마련되어야 파일시스템 영역에서도 물리적인 오류에 대해 적극적으로 대처할 수 있을 것이다. 그래서 FORT를 개발하면서 이와 같은 문제점을 해결할 수 있는 획기적인 오류 검출 도구를 함께 고안하였고 기존 시스템에서도 적용될 수 있는 형태를 갖추었다.

3.1 저장장치에 발생되는 물리적인 오류 탐지 도구 : ERROR-DETECTION

파일시스템 영역과 디바이스 드라이버 영역간의 경계는 명확히 구분되어 있으며 구조적으로 각각의 영역에 대해서 정보교환 인터페이스는 이루어지고 있지 않다. 또한 저장장치에서 물리적인 오류가 발생할 때 에러 레지스터를 검사하는 역할을 디바이스 드라이버 영역에서 독립적으로 다루고 있는 단점을 보완하고자 ERROR-DETECTION모듈을 개발하였다. 파일시스템 영역에서 물리적인 오류를 발견할 수 있는 방법은 없다. 그 이유는 커널 데몬을 통한 디바이스 드라이버 영역으로 요청 큐를 통하여 접근하기 때문에 물리적인 오류가 발생하여 파일시스템에서 에러 레지스터를 검사 하여도 다른 요청 큐에 의해 레지스터 값이 이미 갱신되어 파일시스템 영역에서는 레지스터를 통한 검사를 할 수 없기 때문이다.

ERROR-DETECTION은 저장장치의 오류를 발견하기 위해 디바이스 드라이버 영역의 에러 탐지 함수 내에서 에러 레지스터와 저장장치 관련 레지스터를 검사

표 2 ERROR-DETECTION 모듈의 특징

항 목	설 명
역할	하드디스크 레지스터[14]를 이용해서 저장장치의 물리적인 오류가 발생된 정확한 위치와 오류 구분을 저장 및 관리
자료구조 및 알고리즘	Freelist[15]자 료구조와 제안하는 Free노드 확장 알고리즘
인터페이스	모듈 적재 시 : 저장장치 오류를 발견하고자 하는 저장장치 명 삽입 (예: hda)
	파일시스템 영역 : 오류발생 블록 및 저장장치의 파티션 정보

한다. 그래서 발생된 오류의 위치 및 종류를 자료구조 형태로 관리해주는 모듈이다. 이와 같은 자료구조에 접근 할 수 있는 인터페이스로는 파일시스템에서 구분하는 블록번호와 저장장치에서 구분하는 LBA주소가 모두 가능하도록 구성하였으며 물리적인 오류가 발생한 섹터와 오류의 종류를 쉽게 확인하여 합당한 조치를 취할 수 있도록 하였다.

오류가 발생된 섹터와 오류 종류를 저장하는 자료구조는 성능을 고려하여 메모리 할당과 해제과정을 최소화해야 하고, 저장장치에서 발생되는 물리적인 오류의 개수를 알지 못하기 때문에 노드의 개수에 한계가 있어서는 안된다. 이와 같은 문제점을 해결하기 위해서 일정 개수의 노드를 링크드리스트 형태로 미리 할당 받아 놓고, 물리적인 오류가 발생할 때마다 노드 한 개씩을 가져와 새로운 링크드리스트를 형성하는 것이다. 사용한 노드의 삭제가 필요할 경우 해당 노드에 대한 메모리 해제가 아닌 원래의 리스트로 환원하는 Freelist 자료구조를 모듈에 반영하였다. 하지만 Freelist 자료구조 또한 처음에 할당 받은 리스트의 한계를 가진다. 이를 해결하기 위해서 미리 할당 받은 링크드 리스트의 최소 노드의 개수를 자료구조로 가지고 있다가 이 개수보다 적어졌을 경우 메모리에서 임의의 개수만큼 미리 할당 받아 놓은 링크드리스트의 노드를 늘려주는 알고리즘을 새롭게 제안해서 배열의 장점과 링크드 리스트의 장점을 하나의 자료구조에 접목시켰다.

이와 같은 특성을 가지는 ERROR-DETECTION 모듈을 이용해서 디바이스 드라이버 고유의 워크로드를 파일시스템과 공유함과 동시에 시스템 각 계층 간의 정보교환 인터페이스를 형성할 수 있어 파일시스템의 강인성 및 무결성 그리고 신뢰성까지 보장할 수 있는 지능형 오류 검출 도구로 활용될 수 있다.

3.2 저장장치의 S.M.A.R.T 기능을 활용한 베드섹터 정보 탐지 도구 : KERNEL-SMART

파일시스템 영역에서는 저장장치의 위기상태를 알 수 있는 방법이 없기 때문에 저장장치의 무결성에 문제가 생기면 바로 파일시스템에 영향이 미치게 되어 디지털 정보를 한 순간에 소실될 수 있는 위기 상황에 도달할 수 있다. 따라서 저장시스템의 자가 진단 및 분석을 위해 사용되는 S.M.A.R.T[17]을 이용해서 개발한 KERNEL-SMART 모듈을 커널에 적재 시킨다면 현재의 저장장치의 상태를 즉시 확인할 수 있어 저장장치의 깨진 무결성으로 인한 막대한 디지털 정보 소실을 막을 수 있다. 또한 베드섹터의 발생과 하드디스크의 잉여섹터[16]에 대한 비율 뿐만 아니라 저장장치의 파손 위기 정보를 수치화된 표현으로 모니터링 할 수 있어 ERROR-DETECTION모듈과 함께 커널에 적재 되었을 경우 파일시스템 영역에서 빈번히 발생하는 베드섹터에 대한 상세 정보를 확인할 수 있고 더불어 파일시스템에서 저장장치의 위기에 적극적으로 대처 할 수 있게 된다. 이는저장장치와 파일시스템을 독립적인 계층이 아닌 하나의 유기적인 영역으로 확장할 수 있어 보다 많은 기능을 파일시스템에 부여할 수 있다.

3.3 ERROR-DETECTION 모듈과 KERNEL-SMART 모듈의 상호작용

ERROR-DETECTION 모듈은 저장장치에서 발생되는 모든 물리적인 오류를 레지스터를 통해 발견한 후 본 논문에서 제안하는 자료구조에 저장할 수 있고, KERNEL-SMART 모듈은 디스크의 S.M.A.R.T기능을 이용해서 저장장치의 상태를 언제든지 확인할 수 있는 기능을 가지고 있다. 이 두 가지 모듈을 동시에 커널에 적재 시킨다면 저장장치에서 일어나는 문제에 대한 원인과 발생 위치를 정확히 탐지할 수 있다. 예를 들어 ERROR-DETECTION모듈을 통해서 저장장치의 정확한 위치와 ECC에러임을 확인하였다면 저장장치의 상태를 정확히 검사할 수 있는 KERNEL-SMART모듈을 수행한다. KERNEL-SMART모듈을 통해 사용하지 못하는 섹터가 발견되었다고 확인 되었을 경우 ERROR-DETECTION모듈을 통해 발견된 섹터는 베드섹터임을 명확히 확인 할 수 있게 된다. 이때 해당 섹터는 영구히 사용할 수 없으므로 저장장치가 가지고 있는 잉여 섹터와 재사상[16]을 도와 저장장치의 무결성을 유지할 수 있도록 할 수 있다.

두 모듈은 독립적으로 저장장치의 무결성을 검증할 수도 있지만 동시에 적재하여 서로 융합 하였을 경우 저장장치에서 발생되는 일반적인 오류뿐만 아니라 베드섹터의 발견 및 저장장치를 심각한 상태로 만드는 원인

표 3 KERNEL-SMART 모듈의 특징

항 목	설 명
역할	저장장치의 위기상태 및 베드섹터 관련 상황을 즉시 확인
활용 효과	저장장치의 상태 깨짐과 베드섹터의 발생을 탐지 할 수 있는 효과
극대화 방안	ERROR-DETECTION모듈과 함께 커널에 적재 시 두 모듈의 성능을 극대화 가능
기능	저장장치가 수 시일 내에 파괴될 것을 예상 가능
	베드섹터가 현재 저장장치에 얼마나 발생했는지 파악 가능
	섹터에 대한 재사상[16]으로 인한 하드디스크의 잉여섹터 공간 비율 검사 가능

까지 명확하게 규명할 수 있어 그 기대효과는 극대화 될 수 있다. 따라서 파일시스템뿐만 아니라 리눅스 커널과 저장장치 사이에 계층화된 구조를 하나로 융합할 수 있는 획기적인 제안이며, 동시에 지능형 시스템으로 발전할 수 있는 밑거름이 될 수 있다.

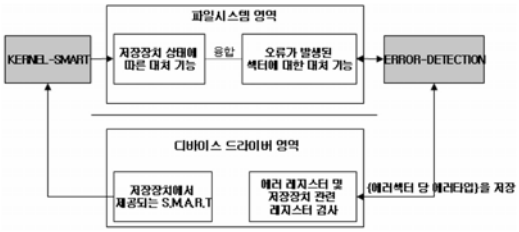

그림 1 ERROR-DETECTION 모듈과 KERNEL-SMART 모듈의 유기적인 상호작용

4. FORT의 구성요소

FORT, 즉 파일시스템의 강인성을 검증 하기 위한 테스트 프레임 워크의 개발은 FORT를 위해 개발된 도구가 있었기 때문에 가능하다. 어플리케이션 계층에서 실제 저장장치 상의 특정 영역에 물리적인 오류를 발생시킬 수 있는 PHYSICAL-FAULT-GENERATOR와 디바이스 드라이버 영역에서 물리적인 오류가 발생한 것과 같은 효과를 발생시켜주는LOGICAL-FAULT-GENERATOR가 있으며, 파일시스템에서 구분하는 저장장치의 어떠한 영역에 읽기/쓰기 접근을 시도하는지를 즉시 확인할 수 있는 실시간 모니터링 도구인 LBAPRINT로 나눌 수 있다. 3가지의 핵심 개발도구를 통해서 FORT를 구성하였으며, 파일시스템의 저장장치 오류에 대한 발견수준과 복구 수준을 표현하기 위해 2가지의 기준표와 최종 결과표로 파일시스템의 강인성과 무결성의 수준을 표현하였다.

4.1 저장장치 상에 실제 베드섹터를 발생시키는 도구 : PHYSICAL-FAULT-GENERATOR

파일시스템 개발자 그룹은 가상의 오류를 디바이스 드라이버 영역에서 발생시킴으로써 테스트 프레임 워크를 진행하고 있으나[9,18] 검증의 신뢰성을 높이기 위해서 실제 저장장치 상에 물리적인 오류를 발생시키고 검증을 진행해야 가장 바람직하다. 이와 같이 실제 저장장치에 물리적인 오류를 발생시키기 위해서는 고의적인 외부 충격으로 저장장치의 어딘가에 발생시켜야 했고 이때의 물리적인 오류가 발생한 정확한 위치를 알 수 없으며 사용자가 원하는 정확한 위치에 발생시킬 수도 없었다. 이와 같은 문제보다 더 큰 문제는 하드웨어적인 결함이 발생할 가능성이 매우 높다는 것이다. 그래서 본 논문에서는 하드웨어적인 결함을 전혀 주지 않으면서 어플리케이션 계층에서 보다 간편하고 정확한 위치 상에 실제 베드섹터를 발생시킬 수 있는 PHYSICAL-FAULT-GENERATOR를 개발하고 검증을 진행하였다.

PHYSICAL-FAULT-GENERATOR 는 기존의 IDE 디스크 에러 발생 도구와의 비교 분석을 통해서 보다 명확하게 특징을 확인하였다. 기존의 저장장치 상에 물리적인 오류 발생 기법을 적용한 SWIFI(Software Implemented Fault Injection)의 대표적인 도구인 IDE 디스크 에러 발생 도구는 디바이스 드라이버 영역에서 사용자가 설정한 특정 섹터 접근 시 가상의 물리적인 오류로 인지하게 만드는 방식을 사용하고 있다. 하지만 PHYSICAL-FAULT-GENERATOR는 읽기 실패를 유도하기 위해 원하는 특정 섹터 상에 가상의 물리적인 오류가 아닌 실제 베드섹터를 발생 시킬 수 있는 큰 차이점을 갖고 있다. 이와 같은 특징으로 기존 도구와는 다르게 정상적으로 저장장치에 접근해서 물리적인 오류를 발생 시킬 수 있어 검증 방식 및 절차에 대한 신뢰성을 향상 시켰다.

기존 SWIFI도구의 경우 커널 자료구조의 수정이 반드시 필요하며, 도구 자체의 자료구조를 저장장치의 입출력이 발생했을 때마다 모두 탐색해야 하기 때문에 성능저하가 반드시 발생한다. 하지만 PHYSICAL-FAULT-GENERATOR는 실제 저장장치 상에 물리적인 오류를 발생 시킬 뿐 어떠한 자료구조와의 연동이 없고 독립적으로 운용되고 있어 파일시스템의 물리적인 저장장치 신뢰성 검증뿐만 아니라 파일시스템의 성능관련 검증에도 상당히 유용하게 활용될 수 있다. 이와 같은 특징으로 인해 SWIFI도구와는 다르게 저장장치의 해당 영역에 대한 디지털 정보가 영구 소실되는 문제점이 있으나 검증 차원에서 보았을 때는 당연한 현상이다.

FORT의 근간이 되고 있는 PHYSICAL-FAULT-

표 4 PHYSICAL-FAULT-GENERATOR와 기존의 IDE디스크에러발생 도구와의 비교분석

비교항목 / 도구	PHYSICAL-FAULT-GENERATOR	기존의 IDE 디스크 에러 발생 도구
특징	디스크 상에 실제 베드섹터 발생	가상의 오류 발생
발생 가능한 오류구분	읽기 실패 (베드섹터)	실패 구분 없이 무조건 오류 적용
시스템 성능의 영향	성능저하 전혀 없음	성능저하 발생 (디바이스 드라이버 영역)
기존 시스템과의 연관성	독립적인 모듈로써 동작	커널 핵심자료구조와 관련 깊음
저장장치 훼손 유무	디지털 정보 소실	디지털 정보의 훼손은 없음

GENERATOR는 기존의 물리적인 오류 발생 도구와는 근본적인 접근 방법이 다르다는 것을 확인 할 수 있었다. 그래서 기존 도구에 대한 비현실성으로 인해 검증 자체의 낮은 신뢰성을 PHYSICAL-FAULT-GENERATOR를 이용함으로써 한층 높일 수 있었다. PHYSICAL-FAULT-GENERATOR는 위의 그림 2에서와 같이 시스템 커널 영역뿐만 아니라 디바이스 드라이버 영역을 모두 접근하지 않은 채 직접 저장장치에 실제 물리적인 오류를 발생시킴으로써 다른 시스템과 연계 없이 독립적인 기능을 갖는 중요한 도구의 특징을 가지고 있다. 이와 같은 특징은 FORT를 위해서 개발된 도구이지만 실제 베드섹터를 발생시키기 때문에 모든 시스템 영역에서 물리적인 오류에 대한 검증을 시도해 볼 수 있는 장점을 가진다. PHYSICAL-FAULT-GENERATOR를 통해 물리적인 오류를 저장장치에 발생시킨 후 다른 시스템에 탑재하더라도 오류 발생 영역에 접근 시 반드시 물리적인 오류가 발생한다. 그래서 위의 그림 2에서와 같이 실제 물리적인 오류에 대한 접근 경로가 일반적인 저장장치의 접근 경로와 같다.

SWIFI와는 다르게 하드디스크 상의 실제 베드섹터를 소프트웨어 계층에서 손쉽고 정확한 위치에 발생시킬 수 있는 도구는 PHYSICAL-FAULT-GENERATOR 뿐이다. 이와 같이 간단한 구조를 가지면서, 실제 저장장치에 접근해서 물리적인 오류를 발생시킬 수 있는 기술은 특정 ATA COMMAND를 이용해서 저장장치에

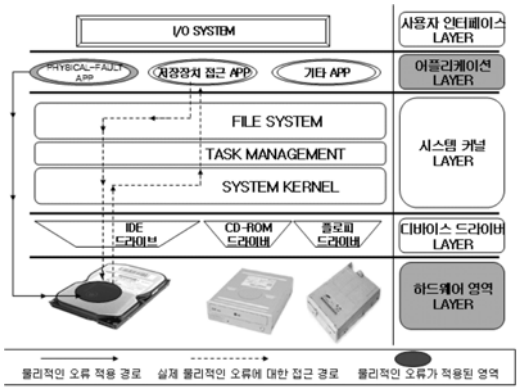

그림 2 PHYSICAL-FAULT-GENERATOR의 물리적인 오류 발생 흐름도

접근 후 물리적인 오류 발생을 위한 ECC 코드를 오염시켰기 때문에 가능하였다. 이와 같은 방식의 물리적인 오류 발생 도구 개발은 최초이며, FORT의 파일시스템 검증뿐만 아니라 하드디스크를 저장장치로 가지고 있는 시스템의 모든 계층에 대한 물리적인 오류 검증을 수행할 수 있어 매우 획기적인 도구가 아닐 수 없다.

이 도구는 리눅스 운영체제 기반에서 개발하였으며, 실제 물리적인 오류를 발생시키고자 하는 저장장치의 선택뿐만 아니라 파티션 단위의 선택이 모두 가능하고, 정확한 섹터 위치를 파일시스템에서 구분하는 블록번호와 저장장치에서 구분하는 LBA주소를 통해 모두 접근이 가능하도록 도구의 인터페이스를 구성하여 파일시스템의 검증을 위한 편리한 구조를 가졌다.

이 도구를 이용해서 원하는 섹터 상에 베드섹터를 발생시켰다면 실제 발생되었는지 확인할 수 있어야 한다. 이를 위해 2가지 방법이 존재한다. 첫째, 파일시스템의 읽기 기능을 이용해서 오류가 적용된 섹터를 읽는다. 만약 실제 물리적인 오류가 적용되었다면 분명히 디바이스 드라이버의 에러관련 함수에서 오류메시지를 보여주게 될 것이다. 둘째, 공개소스로 제공되어지는 S.M.A.R.T도구[17]를 이용해서 Current_Pending_Sector 속성의 RAW_VALUE값 증가로 간단히 확인할 수 있다.

이와 같이 저장장치 상에 베드 섹터를 발생시켰다면 오류가 발생된 섹터를 치료할 수 있는 방법이 있어야 한다. 이 도구는 실제 섹터 상에 베드섹터를 발생시키고 있기 때문에 파일시스템 테스트 프레임 워크를 진행한 후에는 저장장치는 무결성이 깨진 상태가 되어 반드시 베드섹터를 치료할 수 있는 기능이 필요하다. 그래서 저장장치의 베드섹터에 대한 복구 방식인 리매핑[16] 기법을 활용하였다. 이는 베드섹터가 발생된 영역에 더미 데이타를 씀으로써 저장장치의 잉여 섹터와 베드섹터 간의 재사상을 유도하는 방식이다. 물론 더미 데이타를 쓰기 때문에 해당 섹터에 디지털 정보는 영구 소실된다. 이와 같은 복구 기법을 이용해서 PHYSICAL-FAULT-GENERATOR는 ATA COMMAND를 이용한 베드섹터를 발생할 수 있을 뿐만 아니라 섹터 단위의 쓰기가 가능하도록 도구의 기능을 확장하였다. 섹터 단위의 쓰기 기능을 탑재한 이유는 파일시스템의 시스템 콜을 이용하여 베드섹터를 복구할 경우 섹터 단위의 오류가 발

표 5 PHYSICAL-FAULT-GENERATOR 특징 정리

항 목	설 명
특징	어플리케이션 계층에서 저장장치의 특정 섹터상에 베드섹터 발생 및 복구
기법	특정 ATA(Advanced Technology Attachment)[19] COMMAND 활용
활용	베드섹터를 이용하여 파일시스템의 읽기 실패 유도
특이사항	실제 물리적인 오류이므로 해당 섹터의 디지털 정보는 영구 소실

생했음에도 불구하고 블록단위의 쓰기를 수행하여 오류가 발생하지 않은 섹터에도 더미 데이타가 쓰여져서 베드섹터에 인접한 영역까지 디지털 정보의 손실이 발생되는 문제점이 있었기 때문이다.

이처럼 여러 기능을 가진 PHYSICAL-FAULT-GENERATOR는 파일시스템과 저장장치 간의 강인성 연구뿐만 아니라 저장장치의 물리적인 오류에 대한 여러 시스템의 강인성을 연구하는 분야에도 모두 활용될 수 있는 획기적인 도구이다.

4.2 저장장치 상에 가상의 물리적인 오류를 발생시키는 도구 : LOGICAL-FAULT-GENERATOR

기존에 연구 개발되고 있는 물리적인 오류 발생 도구 SWIFI의 하나인 IDE 디스크 에러 발생도구[20]는 오류의 구분 및 각각의 오류에 대한 특성을 반영하지 않은 채 디바이스 드라이버를 수정해서 섹터에 대한 물리적인 오류를 가상으로 발생시키고 있다. 하지만 FORT에서는 저장장치의 원하는 특정 섹터상에 저장장치에서 정의하고 있는 오류[21]를 선택하여 적용시킬 수 있는 특징을 가진 LOGICAL-FAULT-GENERATOR를 개발하여 보다 정확한 파일시스템의 검증을 위해 사용하고 있다. 대표적인 기능으로는 읽기실패, 쓰기실패, 인터페이스 오류, 데이타 오염을 원하는 물리적인 오류 타입과 저장장치의 위치에 적용할 수 있다. LOGICAL-FAULT-GENERATOR는 기존의 IDE 디스크 에러 발생 도구와 같이 디바이스 드라이버 영역을 이용한 물리적인 오류 발생 도구이기 때문에 간단한 비교를 통해서 도구의 우수성을 판단할 수 있으며, 이 도구를 이용해서 파일 시스템 검증 시 우수한 신뢰성을 가질 수 있음을 쉽게 예측할 수 있다.

LOGICAL-FAULT-GENERATOR는 시스템 고유의 성능 저하를 최소화하였으며, 저장장치의 접근 형태에 따른 읽기오류, 쓰기오류, 인터페이스 오류, 데이타 오염의 발생 및 오류에 따른 특징을 반영하여 현실성을 최대한 높여 검증을 위한 신뢰성을 향상시켰다. 그리고 IDE 디스크 에러 발생 도구처럼 배열 자료구조를 사용하여 오류 정보를 담는 형태가 아닌 해쉬 테이블과 링크드리스트 자료구조를 적절히 활용해서 보다 빠른 오류 정보 검색을 유도하였다. 그래서 기존 도구와 같이 전체 배열을 모두 검색해서 시스템이 전체적으로 느려지는 단점을 막고 기존 검증 도구에서는 할 수 없는 여러 가지 형태의 검증을 쉽게 진행 할 수 있게 되었다.

이처럼 빠른 검색을 위한 특징과 여러 접근 형태별 오류발생 및 처리를 할 수 있는 기술적 형태는 필터링 기법을 사용하고 있기 때문이다. LOGICAL-FAULT-GENERATOR에서의 필터링 기법이란, 저장장치의 원하는 영역에 대한 읽기 실패, 쓰기 실패, 인터페이스 오류 구분, 데이타 오염을 두었을 경우 오류 구분에 맞는 형태의 접근이 이루어졌을 때 실제 저장장치의 접근을 막고, 해당 영역에 따라 설정해놓은 물리적인 오류를 발생시키는 형태이다.

그림 3에서와 같이 LOGICAL-FAULT-GENERATOR는 어플리케이션 영역, 시스템 커널 영역을 통해 물리적인 오류를 디바이스 드라이버 영역에서 가상으로 발생시키고 있다. PHYSICAL-FAULT-GENERATOR와 같이 저장장치 상에 실제 물리적인 오류가 발생하는 것이 아닌 디바이스 드라이버 영역에서 물리적인 오류 발생 정보를 통해 실제 저장장치에 접근하지 않은 채 마치 실제 접근해서 물리적인 오류가 발생된 것과 같은

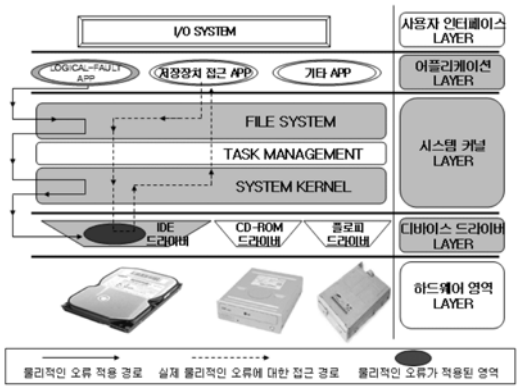

그림 3 LOGICAL-FAULT-GENERATOR의 물리적인 오류 발생 및 오류 접근 흐름도

표 6 LOGICAL-FAULT-GENERATOR와 IDE디스크에러발생 도구와의 비교분석

비교항목 / 도구	LOGICAL-FAULT-GENERATOR	기존의 IDE 디스크 에러 발생 도구
특징	디바이스 드라이버 영역 활용	디바이스 드라이버 영역 활용
저장장치 접근 형태별 오류처리 차별화 유무	차별화된 오류 적용 (읽기오류 / 쓰기오류 / 인터페이스오류 / 데이타 오염)	차별화 없음 (오류 구분 없이 무조건 오류 적용)
시스템 성능의 영향	빠른 검색을 통해 시스템의 최소한의 부하 (해쉬 자료구조 사용)	성능 저하 발생 (배열 자료구조 사용)
기존 시스템의 변형 정도	기존 자료구조의 변형은 없음 (새로운 자료구조 추가)	커널의 핵심자료구조 수정 (Request structure)
저장장치 훼손 유무	훼손 없음	훼손 없음

효과를 주고 있다. 물리적인 오류 적용 경로는 기존의 IDE 디스크 발생 도구와 같다. 하지만 LOGICAL-FAULT-GENERATOR의 특징인 각각의 오류에 따른 필터링 기법이 디바이스 드라이버 영역에서 적용되고 있다는 점이 큰 차이점이다.

예를들어 사용자가 특정 영역에 대한 특정 물리적인 오류를 읽기 실패 구분으로 설정하였다면 설정된 영역에 쓰기 접근이 이루어졌을 때 정상적인 저장장치의 접근이 이루어지고 읽기 접근 시에만 저장장치에 접근하지 않은 채 설정된 물리적인 오류를 발생할 수 있도록 구성하였다. 이와 같은 특징은 기존 도구와 같은 디바이스 드라이버를 이용한 물리적인 오류 발생임에도 불구하고 기존에 사용되는 IDE 디스크 발생도구와는 다른 4가지의 접근 방법에 따른 오류 적용 유무를 반영함으로써 파일시스템의 물리적인 오류에 대한 여러 검증을 손쉽게 진행할 수 있다. 그리고 저장장치에 대한 어떠한 손실도 없는 특징과 물리적 오류에 대한 필터링 부분에서의 지연을 최소화하여 보다 방대한 양의 물리적인 오류를 저장장치에 발생시킬 수 있는 장점이 있다.

물리적인 오류의 특성을 최대한 반영하기 위한 차별화된 물리적인 오류 적용 기법과 LOGICAL-FAULT-GENERATOR의 빠른 필터링 기법을 뒷받침하는 자료구조, 마지막으로 저장장치에 실제 접근하지 않은 채 디바이스 드라이버를 최대한 활용한 LOGICAL-FAULT-GENERATOR의 특징 및 기술적인 환경을 다른 도구와의 비교 및 도구의특징을 통해서 상세히 살펴보았다. 이와 같은 특징을 갖는 LOGICAL- FAULT-GENERATOR는 해당 오류의 성질을 최대한 반영하여 보다 현실성을 높인 것이며 해당 섹터에 접근 하는 방식에 따라 오류의 특성을 반영하여 오류의 적용유무 및 행동을 구분할 수 있도록 한 것은 다른 도구와는 다른 세밀한 도구의 환경을 조성한 것이다.

복구 정책 또한 IDE 디스크 에러 발생 도구에서와 같이 특정 함수만을 이용해서 복구정책이 아닌 차별화된 오류 복구정책을 반영하였다. 각각의 차별화된 접근 방식에 따른 오류 발생 방법과 각각의 접근 방식에 따른 차별화된 복구 기법에 대해서 살펴보도록 하겠다. 첫째, 읽기 오류의 발생과 복구 정책이다. LOGICAL-FAULT-GENERATOR를 이용해서 저장장치의 특정 섹터 위치와 원하는 특정에러 종류 그리고 읽기 오류임을 명시하여 도구를 실행시킨다. 도구를 이용한 읽기 오류가 섹터 상에 반영된 이후 해당섹터에 읽기 수행이 이루어지면, 설정된 에러를 섹터에 적용하여 물리적인 오류를 발생 시킨다. 읽기 오류가 적용된 섹터를 복구하기 위해서는 개발한 복구 함수를 이용해서 오류정보를 삭제 시키는 방법 뿐만 아니라 해당 오류가 적용된 섹터에 쓰기 동작이 반영되었을 경우에도 자동 치료될 수 있도록 하였다. 이는 PHYSICAL-FAULT-GENERATOR에서도 알 수 있듯이 읽기 오류가 발생한 섹터에 쓰기가 수행되면 디지털 정보는 소실되더라도 섹터가 무결하게 복구 될 수 있음을 반영한 것이다. 둘째, 쓰기 오류를 발생하고 싶을 경우 섹터에 대한 읽기는 가능하지만 쓰기 수행 시에 오류를 발생시켜야 한다. 이를 위해서 원하는 섹터와 발생하고자 하는 특정 에러종류 그리고 쓰기 오류임을 명시하여 도구를 실행시킨다. 읽기실패오류와 마찬가지로 설정한 해당 영역에 쓰기 접근이 이루어지면 설정된 물리적인 오류가 발생이 된다. 만약 쓰기 오류를 복구하고 싶다면 개발한 복구 함수를 이용해서 쓰기 오류가 발생한 특정 섹터를 쓰기 오류 리스트에서 해제함으로써 정상적인 쓰기 동작을 수행 할 수 있도록 하였다. 셋째, 인터페이스 오류라 함은 섹터에 접근했을 때 일시적으로 발생하는 오류[22]로써 재 접근 시에는 정상적으로 동작이 수행 되어야 한다. LOGICAL-FAULT-GENERATOR를 이용해서 원하는 섹터 상에 인터페이스 오류를 적용했을 경우 최초접근이 이루어졌을 때 CRC에러[23]를 발생하도록 하였고, 그 즉시 해당 섹터에 설정된 인터페이스 오류를 복구시킴으로써 재 접근 시 섹터에 대한 접근이 정상 작동될 수 있도록 일시적인 오류 현상을 만들어 냈다. 마지막으로 데이타 오염이라 함은 파일시스템에서 특정 데이타를 저장장치에 정상적으로 썼음에도 불구하고 저장장치에는 잘못된 데이타가 쓰여지거나 읽혀지는 것을 의미한다. 이와 같은 환경을 구축하기 위해서 파일시스템에서 저장장치에 특정 데이타를 쓰게 될 때 해당 데이타를 오염시켜 잘못 쓰여지도록 하거나 파일시스템에서 읽으려고 할 때 잘못된 데이타가 반환될 수 있도록 구성하였다. 만약 이와 같이 읽기/쓰기/인터페이스 오류/데이타오염과 같은 특정 행동 구분을 반영하지 않는다면 해당 섹터에 어떠한

표 7 LOGICAL-FAULT-GENERATOR 특징 정리

항 목	설 명
특징	저장장치 상에 접근 형태에 따른 차별화된 물리적인 오류 발생 및 복구
기법	디바이스 드라이버 영역에서 필터링 기법을 이용 후 에러 레지스터[14] 값의 조작
활용	저장장치의 접근 형태에 따른 읽기실패, 쓰기실패, 인터페이스 오류, 데이타 오염 유도
특이사항	가상의 물리적인 오류이므로 해당 섹터의 디지털 정보는 무손실

접근이 이루어지더라도 설정한 특정 오류를 계속 발생하도록 도구를 구성하였으며 사용자가 물리적인 오류가 발생된 섹터를 복구 하고 싶을 경우 언제든지 복구 함수를 통해 복구될 수 있도록 하였다.

이와 같은 특징을 가지고 있는 LOGICAL-FAULT-GENERATOR는 PHYSICAL-FAULT-GENERATOR처럼 물리적인 오류가 아닌 디바이스 드라이버 영역에서 필터링 기법을 사용한 가상의 물리적인 오류이기 때문에 실제 저장장치에는 어떠한 손실도 발생하지 않는 안전한 도구이다. 하지만 PHYSICAL-FAULT-GENERATOR에서 발생한 오류와 같이 실제 저장장치 상에 오류가 적용된 것이 아니기 때문에 오류가 실제 적용되었는지 확인 하는 방법이 제한적이다. PHYSICAL-FAULT-GENERATOR에서 발생한 베드섹터 같은 경우는 S.M.A.R.T 도구를 이용한 검증 방법이 가능하였지만, LOGICAL-FAULT-GENERATOR에서 발생한 저장장치 오류는 실제 저장장치에 적용된 것이 아닌 디바이스 드라이버 영역에서 인위적으로 만든 오류이기 때문에 S.M.A.R.T도구를 이용한 검증은 하지 못하고, 파일시스템을 통해서 오류가 적용된 섹터를 직접 접근하여 디바이스 드라이버 영역에서 보내주는 에러 메시지를 확인하는 방법만을 사용해야 한다. 하지만 PHYSICAL-FAULT-GENERATOR보다 많은 종류의 오류를 발생 시킬 수 있으며 어떠한 환경에서도 자유자재로 손쉽게 만들 수 있는 장점이 있기 때문에 파일시스템을 검증하기 위한 테스트 프레임 워크 구성으로 매우 유용하다.

4.3 저장장치의 읽기/쓰기 접근 상황 모니터링 도구 : LBAPRINT도구 (I/O TRACE TOOL)

리눅스 운영체제 대부분의 파일시스템은 여러 개의 메타 데이타 영역과 데이타 영역으로 구분되어 있으며, 이와 같은 영역을 통해 저장장치에 실제 접근이 시도된다. 따라서 파일시스템과 저장장치의 계층구조 속에서 현재 파일시스템의 어떤 영역에서 저장장치에 접근하고 있으며, 그때의 LBA정보까지 모니터링 되어야 파일시스템의 검증을 위한 세부동작의 시작과 끝을 명확하게 파악할 수 있다. 또한 저장장치에 접근하는 특정 워크로드의 수행 시 파일시스템의 어떠한 영역에서 저장장치에 접근하고 있는지 명확히 확인할 수 있으며 이때 저장장치의 접근하는 위치를 LBA단위로 정확히 확인할 수 있다. 이와 같은 역할을 수행하기 위해 개발된 LBAPRINT 도구는 테스트 프레임 워크를 진행할 때 올바른 저장장치 영역에 접근이 수행되고 있는지 명확히 파악할 수 있고 더불어 테스트 프레임 워크 자체의 신뢰성까지 보장한다.

FORT를 위한 LBAPRINT의 역할 및 기능에 대해서 살펴보았다. 이와 같은 기능을 수행할 수 있는 기술적 알고리즘은 위의 그림 4에서와 같이 파일시스템의 활용과 저장장치상에 접근 시 반드시 디바이스 드라이버를 거친다는 환경에서 구조적인 아이디어를 얻었다. 파일시스템은 블록 단위의 각각의 메타데이타, 저널 영역, 데이타 영역 등의 많은 영역으로 구성되어 있다. 하지만 이와 같은 영역이 실제 저장장치상에 어떠한 영역과 매칭을 이루고 있는지 확인하기는 매우 힘들다. 따라서 일반적으로 위의 그림 4에서와 같이 실선의 화살표 방향으로 저장장치에 접근이 이루어질 때 LBAPRINT는 파일시스템의 자료구조를 통해 현재의 파일시스템이 구분하는 영역별로 로드맵을 구성한 후 각각의 영역에 대해 해당하는 저장장치 상의 정확한 위치를 LBA단위로 확인할 수 있도록 구성하였다.

그림 4에서 볼 수 있듯이 점선 화살표 방향으로 특정 워크로드가 수행 될 경우 해당 워크로드가 파일시스템의 어떠한 영역에서 저장장치 상에 어떠한 영역으로 읽기 또는 쓰기를 수행하고 있는지 한눈에 파악할 수 있다. 아래의 표 8은 LBAPRINT의 특징에 대해서 간략히 정리하였다.

이 모듈은 저장장치에 접근해서 실제 읽기와 쓰기 접근을 시도 하는 작업에 대해서 정확히 모니터링 되야 하기 때문에 모듈이 적재되는 영역은 디바이스 드라이버 내에 실제 ATA COMMAND[19]를 이용해서 저장장치에 읽고 쓰는 부분이어야 한다. 더불어 시스템에서 모니터링 하고자 하는 저장장치와 표 8에서 구분하고 있는 모니터링 항목을 선택하여 모듈을 적재 시킬 수 있다. 또한 검증 절차의 편의성을 도모하기 위해 모듈이 적재 되었을 지라도 특정 오퍼레이션이 수행 될 때만 모듈이 동작할 수 있도록 스위칭 함수를 두어 모듈 동작을 ON/OFF 시킬 수 있도록 하였다.

지금까지 시도되지 않은 이와 같은 형태의 도구는 파

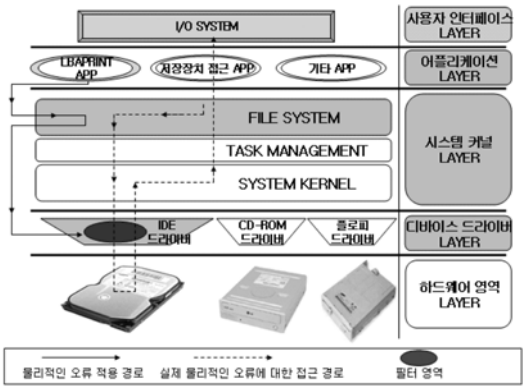

그림 4 LBAPRINT의 시스템 구성도 및 I/O 흐름도

표 8 LBAPRINT도구의 특징 정리

항 목	설 명
역할	파일시스템에서 구분하는 어떠한 영역에서 저장장치에 읽기/쓰기 접근을 시도 하는지 즉시 확인
모니터링 항목	파일시스템의 어떤 동작(읽기/쓰기)을 통해 저장장치에 접근하고 있는가
	파일시스템이 구분하는 영역 중 어느 영역을 통해 저장장치에 접근하고 있는가
	저장장치에 접근 시 정확한 접근 위치(LBA)는 어디인가
기법	파일시스템에서 구분하는 메타 데이타 영역과 데이타 영역 정보를 이용해서 모듈을 디바이스 드라이버 영역에 적용시키고 저장장치에 접근하는 모든 읽기/쓰기 작업을 필터링 하는 기법

일시스템과 저장장치 사이의 주소체계를 보다 투명하게 만들고 파일시스템과 디바이스 드라이버 그리고 저장장치간의 관계를 보다 명확하게 규명할 수 있다는데 큰 의미를 가진 도구이다.

4.4 오류 발생도구와 모니터링 도구의 상호작용

ATA COMMAND[19]를 이용해서 저장장치 상에 실제 베드섹터를 발생 시킬 수 있는 PHYSICAL-FAULT-GENERATOR와 파일시스템에서 저장장치로 접근을 시도 하는 과정 중 중간 영역인 디바이스 드라이버 영역에서 인위적으로 오류를 발생시키는 LOGICAL-FAULT- GENERATOR를 이용해서 저장장치의 원하는 위치에 원하는 물리적인 오류를 발생시킬 수 있음을 살펴보았으며, 파일시스템이 구분하는 어떠한 영역을 통해 어떠한 동작으로 저장장치 상의 물리적인 오류 위치에 접근을 시도 하고 있는지 확인할 수 있는 도구가 LBAPRINT임을 확인하였다. 만약 LBA-PRINT없이 오류 발생 도구를 이용해서 물리적인 오류를 발생시켰고, 원하는 워크로드를 통해 오류 영역에 접근 했더라도 물리적인 오류에 대한 시스템의 반응은 보일 것이다. 하지만 이와 같은 경우에 해당 워크로드가 사용자가 원하는 영역에 접근 했기 때문에 발생한 것인지 아니면 접근하는 다른 영역에 의해 발생된 오류인지를 판단하기 힘들다.

FORT에서는 검증의 신뢰성을 매우 중요시 여겼으며 강조하고 있다. 하지만 이와 같이 불분명한 상황이 존재한다면 FORT에서 강조하는 의미에 위반되는 모습일 것이다. 따라서 2가지 오류 발생 도구를 이용한 오류의 발생에서부터 파일시스템의 어떠한 동작을 통해 오류가 발생된 섹터에 접근하고 있는지 전 과정을 한눈에 파악되야 한다. 이와 같은 취지에 개발된 도구가 LBA-PRINT이다. 만약 오류 발생장치를 이용해서 오류를 발생시키기 전에 LBAPRINT를 탑재했다면 물리적인 오류 영역에 접근 하게 되는 모든 상황을 메시지 형태로써 쉽게 파악할 수 있게 된다. 그래서 그림 5에서와 같이 FORT를 이용한 파일시스템의 검증 시에는 3가지 도구를 시스템에 동시에 적재시킨 후 검증을 시도해야 한다. 그래야 FORT에서 강조하는 신뢰성을 보장할 수 있으며 검증 자체의 편리성을 도모할 수 있다.

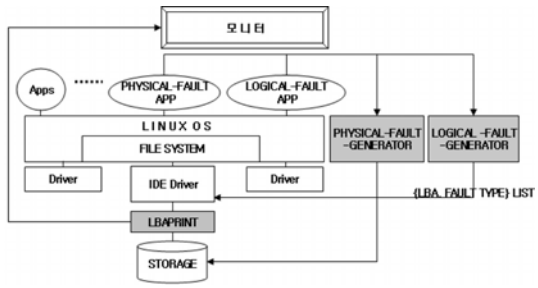

그림 5 시스템 내부에서의 오류 발생 도구와 모니터링 도구간의 상호작용

5. FORT의 진행방법

저장장치의 물리적인 오류에 대한 파일시스템의 테스트 프레임 워크를 구성하기 위해 기존 방식과 다른 핵심 개발도구 3가지의 특징에 대해 상세히 살펴보았다. 이와 같은 개발 도구를 이용하여 본 논문에서 제시하는 테스트 프레임워크를 진행한다면 파일시스템이 저장장치의 물리적인 오류에 대해 얼마나 올바른 대처수준을 가지고 있는지 파악할 수 있어 파일시스템의 강인성 및 신뢰성 수준을 명확히 검증 할 수 있다. 본 논문에서 제시하는 테스트 프레임워크는 4가지의 실패 구분과 2가지의 파일시스템 역할 구분으로 나누고 있다.

2가지의 파일시스템 역할 구분과 4가지의 저장장치 실패 전략을 통해 파일시스템에서 물리적인 오류에 대한 보장 범위를 조사하여 검증하고자 한다. 검증 결과를 표현하기 위한 방법으로 2가지의 기준표를 제안한다. 첫 번째 기준표는 파일시스템이 구분하는 저장장치 영역을 나타내는 표이고, 두 번째 기준표는 저장장치 오류에 대한 파일시스템의 오류 발견 수준 및 복구 수준을 정의

표 9 테스트 프레임 워크의 파일시스템 역할 구분과 저장장치상의 실패 구분표

파일시스템의 역할 구분		저장장치상의 실패 구분
발견	복구	읽기 실패
		쓰기 실패
		데이타 오염
		인터페이스 오류

하는 표로써 각각의 모든 항목을 기호화 하여 간결하게 나타내었다. 이와 같은 2가지 기준표를 이용해서 FORT의 최종 결과표를 작성할 수 있다.

결과표의 구성은 첫 번째 기준표에서 나타낸 파일시스템이 구분하는 저장장치 영역에 따라 파일시스템의 접근 시 오류에 대한 발견 및 복구 수준을 두 번째 기준표에서 확인하여 해당 항목을 FORT의 결과에 기호로써 표시한다. 파일시스템에서 저장장치에 접근을 시도할 때 테스트 프레임 워크 자체의 신뢰성을 향상시키는 일환으로 리눅스 사용자 및 개발자가 주로 사용하는 시스템 콜을 기반으로 워크로드를 구성하였다. 이때 테스트를 진행하기 전에 파일시스템의 함수 호출 흐름을 나타낼 수 있는 환경을 구축한다면 보다 빠르고 정확한 분석이 이루어진다.

5.1 FORT을 위한 환경 구성

데이타를 읽기 위해 저장장치에 접근 시 실패가 이루어지는 것을 읽기 실패, 쓰기를 위해 저장장치에 접근 시 실패가 이루어지는 것을 쓰기 실패, 일시적으로 저장장치의 접근이 실패되는 것을 인터페이스 오류, 읽기실패 및 쓰기실패와는 다르게 데이타의 무결성이 깨진 상태 예를 들어서 메타데이타와 데이타간의 동기화가 맞지 않는 것을 데이타 오염이라 정의하고, 이와 같은 물리적인 오류를 실제 발생시킴으로써 파일시스템의 강인성을 검사할 수 있다.

원하는 형태의 물리적인 오류를 정확한 저장장치의 위치에 발생시키기 위해LBAPRINT를 이용해서 우선 확인한다. 그리고 PHYSICAL-FAULT-GENERATOR 및 LOGICAL-FAULT-GENERATOR를 이용해서 정확한 위치상에 원하는 형태의 오류를 발생 시킨 후 선정된 워크로드 즉, 시스템 콜 기반의 워크로드를 수행한다. 워크로드를 수행 중 물리적인 오류가 발생된 지점을 자연스럽게 접근하게 되면 그때의 파일시스템의 동작과 함수 호출 흐름을 분석함으로써 테스트 프레임 워크를 구성한다. 그리고 보다 현실적이고 정확한 테스트 프레임 워크를 진행하기 위해 쓰기 실패를 제외한 모든 검증에 PHYSICAL-FAULT-GENERATOR를 사용한다.

5.2 FORT의 알고리즘 흐름도

파일시스템의 테스트 프레임 워크를 구성하기 위해 여러 가지 개발 도구와 검증형태에 따른 파일시스템 테스트 프레임 워크의 환경 구성에 대해서 간단히 살펴보았다. 이와 같이 검증형태를 가질 수 있는 이유는 FORT를 위해 개발된 도구들과 파일시스템의 강인성 검증을 위한 알맞은 알고리즘이 개발되었기 때문이다. FORT의 핵심 알고리즘을 이해하기 위해 순차적이면서 단계별 접근방식을 통해 자세히 살펴 볼 것이다. 또한 각 단계별로 절차에 따른 의미를 이해할 것이며 이때 개발된

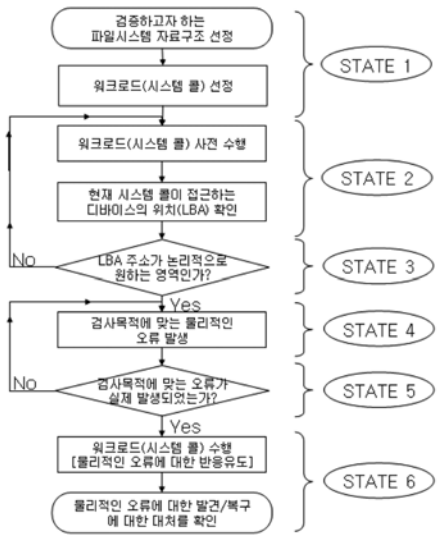

그림 6 파일시스템 테스트 프레임워크 세부 절차 및 단계구분

여러 도구 중에 어떠한 도구가 어떤 방식으로 활용되고 있는가에 대해서도 함께 살펴보겠다.

파일시스템의 여러 자료구조 중 강인성을 검증하고자 하는 영역 또는 자료구조의 선정과 선정된 자료구조에 접근을 시도할 워크로드 즉, 물리적인 오류가 적용된 영역에 접근시킬 시스템 콜을 선정하게 된다. 그리고 LBAPRINT를 시스템에 탑재 된 후 선정된 워크로드를 수행해서 파일시스템의 세부영역 또는 자료구조가 있는 저장장치의 LBA주소를 정확히 파악된다. 그 이유는 도구가 시스템 내에 탑재되어 있어 현재 파일시스템의 어떤 자료구조 및 영역에서 저장장치의 어떤 LBA주소에 접근하고 있는지 파악이 즉시 가능하기 때문이다. 하지만 시스템 콜이 수행 된 후 저장장치에 접근하게 되는 세부영역이 여러 가지 일 수 있다. 여러 가지 영역 중 검증하고자 하는 영역에 해당하는 LBA주소를 확인한다. 하지만 해당 워크로드 수행이 저널이나 캐시에 기록되어 아직 저장장치의 원하는 영역에 접근하지 않을 수 있기 때문에 umount 시스템 콜을 수행해서 데이타를 모두 저장장치에 적용시킨다면 즉시 확인할 수 있다. 만약 저장장치의 원하는 영역에 접근을 하지 않는 경우가 있다면 해당 시스템 콜이 선정된 영역이나 자료구조에 실제 접근하지 않음을 확인해볼 필요가 있다. 이제 확인된 LBA주소를 가지고 쓰기 실패를 제외한 나머지 실패 환경 구축을 위해 PHYSICAL-FAULT-GENERATOR를 이용하고, 쓰기 실패 같은 경우 LOGICAL-FAULT-GENERATOR를 이용해서 정확한 저장장치 위치상에 물리적인 오류를 발생시킨다. 보다 정확한 검

증을 위해서 물리적인 오류를 발생시킨 영역을 실패 구분에 맞추어 간단한 어플리케이션을 통해 접근시켜 본다면 커널 메시지를 통해 오류가 정확히 발생되었음을 즉시 확인할 수 있을 것이다. 지금까지 물리적인 오류에 대한 발견 및 복구수준을 확인하기 위한 환경을 구축하는 단계였다.

이제 선정된 워크로드를 수행시켜 물리적인 오류에 대한 발견과 복구를 실제 하는지, 실제 한다면 발견 및 복구 수준표를 기준으로 어떠한 수준을 갖고 있는지를 조사한다. 이때 파일 시스템 영역에서 입출력 흐름을 확인할 수 있도록 환경을 구축하고 진행한다면 보다 빠른 검증을 수행할 수 있다. 이와 같은 FORT의 세부 절차에 대해서 단계별로 명확한 의미와 이때 참조되는 표 및 개발 도구를 나타내었다.

FORT는 위의 표 10에서와 같이 6단계의 중요한 절차 속에서 FORT를 위해 고안된 도구 또는 표를 이용해서 파일시스템 테스트 프레임 워크를 손쉽게 진행할 수 있다. 그리고 각각의 자료구조와 세부 영역까지 검증하기 위해 기존엔 복잡한 과정과 저장장치 상에 실제 물리적인 오류가 아닌 가상의 물리적인 오류를 발생시켰기 때문에 검증 자체의 신뢰성을 의심해볼 필요가 있었다. 하지만 FORT에서는 검증을 위한 도구에서부터 신뢰성과 현실성을 가지고 있으며 워크로드를 일반 사용자들이 많이 사용하는 시스템 콜로 하였기 때문에 검증의 신뢰성을 한층 높일 수 있었다. 이와 같은 특징을 이용해서 FORT의 핵심 알고리즘 및 세부 절차에 대해서 알아보았다.

만약 원하는 특정 파일시스템에 대해 FORT를 이용한 검증을 진행한다면 해당 파일시스템이 물리적인 오류에 대한 자신이 사용하고 있는 파일시스템의 우수성 또는 심각성을 쉽게 알 수 있는 계기가 될 것이다. 그리고 파일 시스템뿐만 아니라 다른 시스템 개발자들도 자신이 개발하는 시스템의 강인성을 향상시킬 수 있는 방향을 제시할 수 있는 기본 틀을 제공하는 것이다.

5.3 FORT의 단계별 적용 사례

지금까지는 FORT를 위해서 개발된 도구들과 이 도구를 활용한 FORT의 세부 알고리즘을 순서도와 단계별 절차 및 의미를 통해 자세히 살펴보았다. 하지만 시스템 커널의 일부 영역인 파일시스템은 어플리케이션과는 다르게 검증하기 위해서 많은 배경 지식과 검증하기 위한 절차에 대해서 명확히 숙지해야만 검증 결과에 대한 신뢰성을 갖출 수 있다.

그래서 본 절에서는 지금까지 FORT를 위해 제시된 세부 알고리즘을 바탕으로 단계별 적용 방법을 읽기 실패와 쓰기 실패에 대해서 적용 사례를 보이도록 하였다. 파일시스템이 물리적인 오류로 인한 읽기 실패와 쓰기 실패가 발생하였을 경우 어느 수준의 강인성을 갖고 있는지 검증하기 위해 mount 시스템 콜을 적용사례로 선정하였다. 또한 FORT에 대한 세부 알고리즘 이해를 돕기 위해 FORT의 단계별 절차 및 의미표를 통한 순차적 접근 방법을 사용하였다.

STATE1에서 수행하고자 하는 워크로드 즉, 시스템 콜을 결정하고, 해당 시스템 콜이 접근하는 자료구조 중 검증하고자 하는 영역을 선정하게 된다. 그래서 STATE 1에서는 Mount 시스템 콜을 이용해서 해당 콜이 접근하는 여러 자료구조 중 inode table 영역의 읽기 실패에 따른 검증을 결정하였고, super block 영역의 쓰기 실패에 따른 검증을 결정하여 물리적인 오류에 대한 발견 및 복구 수준을 나타내기로 하였다. STATE2에서 Mount시스템 콜을 LBAPRINT와 함께 수행시켜 접근하는 영역에 대한 LBA주소를 확인한다. 이때 검증하고자 하는 영역의 LBA주소만 나타날 수도 있겠지만 대다수의 경우 해당 시스템 콜이 접근하는 모든 영역에 대해서 저장장치의 위치를 알려주게 된다. 이때 STATE3에서와 같이 검증을 원했던 영역의 LBA주소만을 확인한다. 읽기 실패 및 쓰기 실패 시 모두 LBAPRINT에

표 10 FORT의 단계별 절차 및 의미

STATE	절차 별 의미	적용 도구 및 참조표
STATE 1	저장장치의 물리적인 오류발생을 적용시킬 영역선정 물리적인 오류 발생 영역 접근 워크로드 선정	표 9, 표 14
STATE 2	선정된 워크로드가 접근하는 파일시스템의 세부영역과 위치(LBA) 확인	Lbaprint
STATE 3	워크로드가 접근하는 파일시스템의 영역 중 검증하고자 하는 영역에 대한 저장장치의 위치(LBA) 파악	원하는 검증 영역에 대한 위치 확인
STATE 4	읽기/쓰기/데이타오염/인터페이스 오염 중 원하는 물리적인 오류를 도구를 이용해서 발생	PHYSICAL-FAULT-GENERATOR LOGICAL-FAULT-GENERATOR
STATE 5	물리적인 오류가 정확한 위치에 발생되었는지 확인	실패 구분에 맞게 해당 영역을 실제 접근하여 물리적인 오류 발생 확인
STATE 6	워크로드(시스템 콜)수행 후 파일시스템의 물리적인 오류에 대한 발견 및 복구 수준 조사	파일시스템 반응 및 I/O 흐름 조사

서 inode table 영역에 대한 정확한 위치와 super block 영역에 대한 정확한 위치를 메시지 형태로 확인할 수 있다. 따라서 Mount 시스템 콜에서 접근하는 inode table의 저장장치 위치를 LBA 주소 95로 확인 할 수 있었으며, Super block에 쓰기 접근을 수행하기 위해서는 LBA 63에 쓰기 수행함을 확인할 수 있었다. 파일시스템의 검증을 위해 탐색된 위치정보를 통해 읽기 실패 환경을 만들 수 있는 PHYSICAL-FAULT-GENERATOR, 쓰기 실패 환경을 만들 수 있는 LOGICAL-FAULT-GENERATOR를 통해 inode table이 위치하게 되는 LBA 95에 읽기 실패, super block이 위치하게 되는 LBA 63에 쓰기 실패가 발생하도록 환경을 구축하는 것이 STATE 4이다. STATE 5에서는 실제 읽기 실패와 쓰기 실패가 올바르게 발생되었는지 확인하기 위해서 간단하게 읽기 실패가 발생된 부분을 읽기 시스템 콜을 이용해서 읽고, 쓰기 실패된 부분에 대해서는 쓰기 시스템 콜을 이용해서 써본다면 실제 물리적인 오류가 발생되었음을 확인할 수 있다.

STATE 4까지는 저장장치상에 실제 물리적인 오류 발생 및 검증 환경을 위한 단계라면 STATE 5는 실제 선정된 워크로드였던 mount 시스템 콜을 수행시키는 단계이다. 이때 읽기 오류를 발생시킨 inode table과 쓰기 오류를 발생시킨 super block에 대한 접근이 이루어지면 파일시스템 및 디바이스 드라이버 영역에서 출력되는 메시지를 확인할 수 있고 커널 소스의 간단한 확인 절차를 마지막으로 발견 수준과 복구 수준에 대해서 정의 할 수 있다.

읽기 실패가 이루어지면 표 12에서와 같은 메시지를 확인할 수 있으며 메시지를 통한 커널 소스의 접근을 시도할 수 있다. 각각의 메시지를 통해서 물리적인 오류에 대한 파일시스템의 반응을 확인할 수 있으며 위의 메시지를 통해 LBA 95에 물리적인 오류가 발생되어 해당 섹터를 읽지 못한다는 커널 메시지를 확인할 수 있다. 그리고 파일 시스템영역에서는 inode table에 읽기 실패가 발생하여 해당 영역을 읽지 못함을 알려주고 있다. 파일 시스템의 커널 메시지를 보면 에러처리를 위해서 수행되는 ext3_get_inode_loc함수를 끝내지 않고 ext3_handle_error 함수를 호출하여 에러 처리 과정에 진입함을 커널 소스 레벨에서 간단히 확인 할 수 있었다. 따라서 mount 시스템 콜은 inode table 접근 시 하위 영역에서 검사가 가능한 에러 코드를 리턴 받아 물리적인 오류를 발견하고 있음을 확인하였다. 그리고 사

표 11 mount 시스템 콜의 읽기/쓰기 실패에 대한 검증 절차

STATE	읽기 실패에 대한 검증 절차	쓰기 실패에 대한 검증 절차
STATE 1	mount system call이 접근하는 여러 영역 중 inode table에 대한 검증 결정	mount system call이 접근하는 여러 영역 중 super block에 대한 검증 결정
STATE 2	LBAPRINT를 탑재한 후 시스템 콜을 수행한 후 검증 영역에 해당하는 LBA주소를 확인	LBAPRINT를 탑재한 후 시스템 콜을 수행한 후 검증 영역에 해당하는 LBA주소를 확인
STATE 3	Mount 시스템 콜이 접근하는 inode table에 대한 LBA주소 확인(LBA주소 : 95)	Mount 시스템 콜이 접근하는 super block에 대한 LBA주소 확인(LBA주소 : 63)
STATE 4	LBA 95번지에 PHYSICAL-FAULT-GENERATOR를 이용해서 물리적인 읽기오류를 발생.	LBA 63번지에 LOGICAL-FAULT-GENERATOR를 이용해서 가상의 물리적인 쓰기오류를 발생.
STATE 5	LBA 95번지에 읽기 시스템 콜을 이용해서 실제 에러 메시지 확인	LBA 63번지에 쓰기 시스템 콜을 이용해서 실제 에러 메시지 확인
STATE 6	실제 mount시스템 콜을 수행해서 파일시스템의 반응 및 I/O 흐름 조사	실제 mount시스템 콜을 수행해서 파일시스템의 반응 및 I/O 흐름 조사

표 12 mount 시스템 콜의 읽기 실패 시 확인 메시지

	읽기 실패가 이루어졌을 경우 확인 할 수 있는 메시지
커널 출력 메시지	hdb: dma_intr: status=0x51 {DriveReady SeekComplete Error} hdb: dma_intr: error=0x40 {UncorrectableError}, LBAsect=95, high=0, low=95, sector=32 end_request: I/O error, dev 03:41 (hdb), sector 32
파일 시스템 메시지	Ext3_get_inode_loc: unable to read inode block - inode=8, block=4 mount: wrong fs type, bad option, bad superblock on /dev/hdb1, or too many mounted file system ext3-fs error (device ide0(3,65)): ext3_get_inode_loc: unable to read inode block - inode=8, block=4
간단한 커널 소스 확인	…생략… START FUNCTION : ext3_handle_error START FUNCTION : ext3_error_behaviour …생략…

표 13 mount 시스템 콜의 쓰기 실패 시 확인 메시지

	읽기 실패가 이루어졌을 경우 확인 할 수 있는 메시지
커널 출력 메시지	hdb: dma_intr: status=0x51 { DriveReady SeekComplete Error } hdb: dma_intr: error=0x40 { UncorrectableError }, LBAsect=70, high=0, low=70, sector=0 end_request: I/O error, dev 03:41 (hdb), sector 0
파일 시스템 메시지	EXT3 FS 2.4-0.9.19, 19 August 2002 on ide0(3,65), internal journal
간단한 커널 소스 확인	…생략… END FUNCTION : ext3_commit_super …생략…

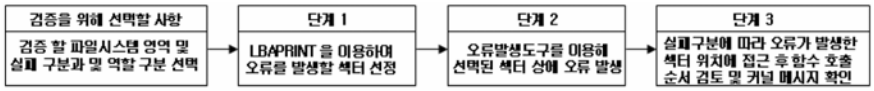

그림 7 테스트 프레임 검증순서 블록 다이어그램

용자에게 현재 inode table의 읽기 실패가 이루어져 mount가 수행되지 않았음을 메시지를 통해 확인시킨 후 mount 과정이 중단되었음을 알려주고 있다.

읽기 실패와 마찬가지로 쓰기 실패가 이루어지면 표 13과 같은 메시지를 확인할 수 있으며, 메시지를 통한 커널 소스의 접근을 시도할 수 있다. 우선 물리적인 오류가 발생하였기 때문에 커널 영역에서 해당 위치와 에러 구분이 담긴 커널 메시지를 보여주고 있음을 확인할 수 있다. 그러나 읽기실패에서와는 다르게 파일시스템 영역에서의 쓰기 실패에 따른 에러 메시지는 확인 할 수가 없었다. 커널의 디바이스 드라이버 영역에서는 물리적인 오류로 인한 에러메시지를 확인할 수 있었지만 파일시스템 영역에서는 어떠한 에러 메시지를 확인할 수 없었으며 시스템 콜 또한 실패를 반환하지 않는 것으로 봐서는 물리적인 쓰기 실패에 대해서는 파일시스템이 어떠한 고려도 하지 않음을 알 수 있다. 마지막으로 커널 소스 레벨에서 확인한 결과 mount 동작 시 ext3_commit_super는 super block에 대한 내용을 갱신시키기 위해서 쓰기 작업을 수행하지만 쓰기 실패에 대한 어떠한 고려도 하지 않음을 확인함으로써 보다 명확해졌다. 따라서 mount 시에 super block영역에 대한 쓰기 오류의 발견은 어떠한 발견 수준을 갖지 않음을 확인하였으며 복구 수준 또한 어떠한 처리도 하지 않는다는 결론을 낼 수 있었다.

지금까지 읽기 실패와 쓰기 실패에 대한 간단한 적용사례를 보면서 정확한 FORT의 알고리즘과 상세한 단계별 절차에 대해서 살펴보았다. 그리고 단계별 절차가 의미하는 것이 무엇이며, 각각의 절차를 통해서 얻어질 수 있는 정보에 대해서도 함께 살펴보았다.

6. FORT을 이용한 파일시스템 검증

FORT의 구성을 위해 개발된 도구와 논문에서 제시하는 저장장치의 4가지 실패 전략 및 2가지 파일시스템 역할 구분을 바탕으로 실제 파일시스템에 적용하여 강인성과 무결성을 검증하고자 한다. 검증의 대상이 될 파일시스템은 신뢰성 및 일관성을 최상위 수준으로 유지하기 위해 저널링 기능을 이용하는 EXT3 파일시스템으로 선정하였다. 선정된 파일시스템뿐만 아니라 개발되어진 파일시스템 전부를 테스트 프레임워크에 적용할 수 있지만 신뢰성 및 일관성을 최상위 수준으로 유지하는 파일시스템이기 때문에 과연 물리적인 저장장치와 묶여있는 관계 속에서 얼마나 견고하고 강인한지 검증을 위해 선정하였다.

6.1 검증 순서

사용자가 선택한 오류 구분에 따라 물리적인 오류를 발생 시켜 파일 시스템에서 정확하게 오류를 발견하고 복구를 수행하고 있는지 위와 같은 단계를 거쳐 검증을 진행한 후 결과표를 작성 할 수 있다. 파일시스템 영역에서 오류섹터에 접근 시 물리적인 오류의 발견수준을 검증할 지 복구수준을 검증할지를 결정하고 4가지의 실패구분 중 하나를 결정하여 각각 검증을 진행한다. 개발도구 중 LBAPRINT를 이용해서 현재 저장장치 상에 파일시스템이 나누고 있는 영역 중 원하는 영역의 LBA 주소를 알아내고 전략에 맞는 오류 발생 도구를 선택하여 해당 LBA주소 상에 물리적인 오류를 발생시킨다. 이제 오류구분에 따라 오류가 발생된 섹터에 접근해서 오류에 대한 파일시스템의 함수 호출 흐름과 행동을 관찰하여 오류 발견과 오류 복구 수준을 검증 한다.

6.2 검증 결과

테스트 프레임 워크를 EXT3파일시스템에 적용하여 결과표를 작성하였고, 하드디스크 에러에 대한 오류 발견 수준과 복구 수준을 검증하기 위해 가장 흔히 발생되는 읽기 실패와 쓰기 실패에 대해 검증하였다.

테스트 프레임 워크 자체의 신뢰성을 위해 개발자 및 응용 프로그램에서 주로 사용하는 시스템 콜을 바탕으로 검증 항목을 구성 하였으며, 파일시스템에서 나누고

표 14 파일시스템이 구분하는 저장장치의 영역

기 호	영 역 구 분	기호	영 역 구 분
a	Inode bitmap	g	g descriptor
b	Inode table	h	j super
c	Data bitmap	i	j revoke
d	Data block	j	j descriptor
e	Data indirect	k	j commit
f	Super block	l	j data

표 15 오류발견 및 복구 수준 적용 문자표

기 호	발견 수준
α	에러를 발견하기 위한 어떠한 작업도 하지 않음
β	하위영역으로부터 전달 받아 에러를 발견
기 호	복구 수준
α	에러를 복구하기 위한 작업을 하지 않음
β	현재의 작업을 중단하는 복구방법
ɣ	현재의 에러를 사용자에게 알려주는 복구방법
δ	β와ɣ의 복구동작을 동시에 수행하는 방법
ε	ɣ 와 현재의 작업을 반복하는 복구 방법

표 16 읽기 실패의 경우 오류 발견 수준표

작 업	a	b	c	d	e	f	g	h	i	j	k	l
Creat	β	β	β			β						
Read		β		β	β							
Write			β	β	β							
Link		β	β									
Unlink	β	β	β	β	β							
Mkdir	β	β	β			β						
Rmdir	β	β	β	β								
Mount		β				β	β	β				
Umount						β		β				

표 17 읽기 실패의 경우 오류 복구 수준표

작 업	a	b	c	D	e	f	g	h	i	j	k	l
Creat	δ	δ	δ			δ						
Read		δ		ɣ	ε							
Write			β	ɣ	α							
Link		δ	δ									
Unlink	α	δ	α	ɣ	α							
Mkdir	δ	δ	δ			δ						
Rmdir	α	δ	α	ɣ								
Mount		δ				δ	δ	δ				
Umount						δ		δ				

있는 저장장치의 영역별로 오류의 발견과 복구 수준을 기호화 하여 표로 작성하였다. 표 내에 어떠한 기호도 적용되지 않은 공간은 해당 시스템 콜이 수행하는 과정에서 해당 저장장치 영역에 어떠한 접근도 하지 않기 때문이다. 2가지 결과표의 가로축은 파일시스템 영역에서 구분하는 세부 영역으로써 저장장치를 표현하였으며, 세로축은 시스템 콜로써 파일시스템을 표현하였다. 이와

표 18 쓰기 실패의 경우 오류 발견 수준표

작 업	a	b	c	d	e	f	g	h	i	j	k	l
Creat	α	α	α		α	α	α	α		β	β	β
Read		α						α		β	β	β
Write		α	α	α	α		α	α		β	β	β
Link		α	α		α		α	α		β	β	β
Unlink	α	α	α	α	α		α	α	β	β	β	β
Mkdir	α	α	α		α	α	α	α		β	β	β
Rmdir	α	α	α	α	α		α	α	β	β	β	β
Mount						α						
Umount	α	α	α	α	α		α					

표 19 쓰기 실패의 경우 오류 복구 수준표

작 업	a	b	c	d	E	f	g	H	i	j	k	l
Creat	α	α	α		α	α	α	α		δ	δ	δ
Read		α						α		δ	δ	δ
Write		α	α	α	α		α	α		δ	δ	δ
Link		α	α		α		α	α		δ	δ	δ
Unlink	α	α	α	α	α		α	α	δ	δ	δ	δ
Mkdir	α	α	α		α	α	α	α		δ	δ	δ
Rmdir	α	α	α	α	α		α	α	δ	δ	δ	δ
Mount						α						
Umount	α	α	α	α	α		α					

같은 결과표를 이용해서 시스템 콜이 접근하는 저장장치 영역에 대한 물리적인 오류의 발견 수준표와 복구 수준표를 작성하였다.

6.3 결과 분석

검증을 수행한 Ext3 파일시스템에서 디스크를 읽기 수행 시 물리적인 오류로 인한 2가지 분석표를 통해 파일시스템의 강인성을 쉽게 분석할 수 있다. 테스트 프레임워크를 적용한 모든 시스템 콜은 저장장치에 접근 시 물리적인 오류에 대해 에러코드를 돌려줌으로써 오류를 발견하고 있어 "β"와 같은 발견수준을 적용하였다. 복구표를 통해서는 매우 다양한 복구수준이 적용되고 있음을 한눈에 알 수 있다. 예를 들어 읽기 시스템 콜과 마운트 시스템 콜을 분석해 보면, 읽기 시스템 콜인 경우 Inode table, Data block, Data indirect 영역에 각각 접근한다. Inode table에 접근 시 저장장치에서 오류가 발견이 되면 현재의 작업을 중단하고, 현재의 오류 상태를 사용자에게 알려주는 방식의 오류에 대한 복구를 진행하며, Data block에 접근 시에 저장장치 오류가 발견이 되면 현재의 오류 상태를 사용자에게 알려주는 방식만을 취하고 있다. 마지막으로 Data indirect영역은 테스트 프레임 워크 결과에서 유일하게 현재의 오류 상태를 사용자에게 알려주며 재 접근을 시도하고 있다. 그래서 δ, ɣ, ε의 복구 수준을 각각 적용시켰다. 마운트 시스템 콜인 경우 Data bitmap, Super block, g-descriptor,

j-super영역에 각각 접근하며 저장장치에서 오류가 발견이 되면 현재의 작업을 중단하고 오류 상태를 사용자에게 알려주는 복구 방법을 사용하고 있음을 한눈에 파악할 수 있다.

저장장치의 쓰기 수행 시 물리적인 오류로 인한 2가지 분석표를 통해 파일시스템의 강인성을 쉽게 분석 할 수 있다. 쓰기 실패에 대한 오류 발견 수준표를 보았을 때 대부분의 시스템 콜이 j-super 영역을 제외한 저널 영역에서 물리적인 오류에 대한 에러코드를 통해 오류를 발견하고 있다. 따라서 "β" 발견수준을 적용하였고, 그 외의 접근하는 영역에 대해서는 오류를 발견하기 위한 어떠한 작업도 하지 않아 "α"발견 수준을 적용하였다. 저장장치 오류에 대한 복구 수준표와 발견 수준표는 비슷한 형태를 보이고 있다. j-super 영역을 제외한 저널 영역에 대해서는 저장장치에 접근 시 물리적인 오류에 대해서 현재의 작업을 중단하고 현재의 오류 상태를 사용자에게 알려주는 복구 방법을 사용해서 "δ" 복구 수준을 적용하였으며 나머지 영역에 대해서는 오류를 복구하기 위한 어떠한 작업도 하지 않아 "α"복구 수준을 적용하였다. 이와 같이 EXT3 파일시스템의 강인성 테스트를 위한 프레임워크 분석표를 작성하고, 분석까지 완료하였다. 분석표를 통해 저장장치와 얽혀 있는 유기적인 관계 속에서 얼마나 강인성을 보장하고 있는지 쉽게 파악할 수 있었다.

검증을 진행한 EXT3 파일시스템은 읽기 수행 시 물리적인 오류 정보를 발견하더라도 어떠한 복구 동작도 하지 않거나 사용자에게 오류에 대한 정보만을 알려주는 경미한 복구 수준을 가지고 있음을 확인하였다. 쓰기 수행 시에도 j-super영역을 제외한 저널 영역에서만 오류를 발견할 뿐 다른 영역에서는 오류에 대한 어떠한 발견도 하지 않고 있다. 복구 동작 또한 j-super영역을 제외한 저널 영역에서만 현재의 작업을 중단하는 복구 동작이 있을 뿐 대부분의 영역에서는 어떠한 복구 동작도 하지 않는 EXT3 파일시스템의 문제점을 파악할 수 있었다.

이와 같은 최종 결과를 통해 실제 EXT3 파일시스템이 운용되는 도중 심각한 문제를 일으킬 수 있는 경우를 생각해 볼 수 있었다. 파일시스템이 저장장치에 적재되기 위해 마운트 동작을 수행한다면 슈퍼블록 영역에 어떠한 저장장치 오류가 있다 하더라도 오류에 대한 어떠한 대처도 하지 않기 때문에 신뢰성 및 견고성을 위주로 개발된 EXT3저널링 파일시스템에 문제점을 증명할 수 있었다.

7. 결론 및 향후 연구

대용량화 되어 가는 저장장치와 파일시스템과의 강인성과 신뢰성을 검증할 기존의 프레임워크[24]는 명백한 한계점이 있다. 기존에 존재하는 테스트 프레임 워크는 신뢰성 및 무결성을 위한 테스트 도구로써 대부분 파일시스템의 논리적인 오류와 견고성만을 검증하고 있다. 따라서 기존 파일시스템 프레임워크와 접근방식이 완전히 다른 새로운 개념의 테스트 프레임 워크를 제안하였다. 제안된 테스트 프레임 워크는 4가지의 실패 구분에 따른 파일시스템의 2가지 역할 구분으로 나누었으며, 이를 위해 핵심 개발도구 3가지를 소개하였다. 또한 신뢰성 및 일관성을 최상위 수준으로 유지하는 EXT3 파일시스템에 실제 적용하여 결과표를 작성하고 문제점을 찾아보았다. 다른 많은 파일시스템에 대해서 이와 같은 테스트 프레임 워크를 적용한다면 파일시스템의 강인성에 대한 문제점을 파악할 수 있고 향후 많은 보완이 이루어질 것으로 예상 할 수 있다.

이와 같은 문제점이 발생하는 가장 큰 이유가 디바이스 드라이버 영역과 파일시스템 영역이 명백히 나뉘어져 있기 때문이다. 따라서 두 영역을 서로 융합할 수 있는 ERROR-DETECTION 모듈과 KERNEL-SMART 모듈을 개발하고 소개하였다. 2가지 개발된 모듈을 시스템에 적재한다면 파일시스템영역에서 저장장치 자체의 물리적인 결함 및 심각한 문제를 빠르게 파악할 수 있고, 디지털 정보의 심각한 영향을 주는 베드섹터에 대한 실시간 상황파악이 가능해지기 때문에 디지털 정보를 적극적으로 보호 할 수 있다. 향후 기존에 개발 되어진 파일시스템보다 견고하고, 논리적/물리적인 오류에 강인한 파일시스템이 개발된다면 본 논문에서 제안하는 테스트 프레임워크를 적용하여 강인함을 증명할 계획이다.

참 고 문 헌

[1] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Model-Based Failure Analysis of Journaling File Systems," presented at Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05) Brighton, United Kingdom, 2005.

[2] S. Gopalan, P. W. Charles, and Z. Erez, "Ensuring data integrity in storage: techniques and applications," presented at Proceedings of the 2005 ACM workshop on Storage security and survivability, Fairfax, VA, USA, 2005.

[3] Linux Test Project, "http://ltp.sourceforge.net".

[4] "The Data Clinic, Hard Disk Failure, http://www.dataclinic.co.uk/hard-disk-failures.htm," 2004.

[5] T. J. Kowalski, "Fsck : the UNIX file system check program," in UNIX Vol. II: research system (10th ed.): W. B. Saunders Company, 1990, pp. 581-592.

[6] Mkfs-build a linux file system, "http://linux.about.com/od/commands/l/blcmdl8_mkfs.htm".
[7] Tripwire Inc. Tripwire Software. "http://www.tripwire.com".
[8] H. K. Gene and H. S. Eugene, "The design and implementation of tripwire: a file system integrity checker," presented at Proceedings of the 2nd ACM Conference on Computer and communications security, Fairfax, Virginia, United States, 1994.
[9] P. Vijayan, N. B. Lakshmi, A. Nitin, S. G. Haryadi, C. A.-D.Andrea, and H. A.-D. Remzi, "IRON file systems," presented at Proceedings of the twentieth ACM symposium on Operating systems principles (Brighton, United Kingdom, October 23 - 26, 2005), Brighton, United Kingdom, 2005.
[10] hard disk constitution, "http://www.pcguide.com/ref/hdd/op/index.htm".
[11] K. Hannu, S. Heikki, and L. Fabrizio, "Detection of Defective Media in Disks," presented at Defect and Fault Tolerance in VLSI Systems, 1993., TheIEEE International Workshop on, Venice, Italy, 1993.
[12] K. Hannu, S. Heikki, and L. Fabrizio, "Detecting Latent Sector Faults in Modern SCSI Disks," presented at Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS '94., Proceedings of the Second International Workshop on, Durham, NC, USA, 1994.
[13] hard disk device drive, "http://www.storagereview.com/guide2000/ref/hdd".
[14] Hard disk register & ide, "http://hem.passagen.se/communication/ide.html".
[15] Freelist Data Structure. "http://www.graphviz.org/pub/graphviz/CURRENT/doxygen/html/structfreelist.html".
[16] Sector remapping, "http://www.storagereview.com/guide2000/ref/hdd/perf/qual/featuresRemap.html".
[17] S.M.A.R.T., "http://www.die.net/doc/linux/man/man8/smartd.8.html".
[18] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek, "Fault Injection Experiments Using FIAT," IEEE Trans. Comput. 39, 4 (Apr. 1990), vol. 39, pp. 575-582, 1990.
[19] The Guide to ATA/ATAPI documentation. "http://www.stanford.edu/~csapuntz/ide.html".
[20] Jake Adriaens, Dan Gibson, "A Software Layer for IDE Disk Fault Injection", System Lacking Originality Workshop 2005.
[21] EXT3 File system harddisk register value. "/usr/src/linux/include/linux/hdreg.h".
[22] A. Jean, A. Martine, A. Louis, C. Yves, F. Jean-Charles, L. Jean-Claude, M. Eliane, and P. David, "Fault Injection for Dependability Validation: A Methodology and Some Applications," IEEE Trans. Softw. Eng. %@ 0098-5589, vol. 16, pp. 166-182, 1990.
[23] Intel reporting data. Serial e05133r0 "Serial ATA ICRC Reporting".
[24] J. Yang, P. Twohey, D. Engler, and M. Musuvathi, "Using Model Checking to Find Serious File System Errors," presented at Sixth Symposium on Operating Systems Design and Implementation, 2004.

김 영 진
2005년 평택대학교 컴퓨터과학과 학사 2005년 8월~현재 한양대학교 공과대학 전자컴퓨터통신공학부 석사 재학중. 관심분야는 파일시스템, 임베디드 시스템, 멀티미디어 시스템, 운영체제

원 유 집
1990년 서울대학교 자연과학대학 계산통계학과 학사. 1992년 서울대학교 자연과학대학 계산통계학과 석사. 1997년 University of Minnesota 전산학 박사. 1997년~1999년 Performance Analyst, Intel Corp. 1999년 3월~현재 한양대학교 공과대학 전자컴퓨터통신 공학부 부교수. 관심분야는 멀티미디어 시스템, 운영체제, 컴퓨터 네트워크

김 락 기
2006년 부경대학교 전자통신컴퓨터공학과 학사. 2006년~현재 한양대학교 공과대학 전자컴퓨터통신공학부 석사 재학중. 관심분야는 운영체제, 파일시스템, 임베디드 시스템

이 모 원
2005년 남서울대학교 정보통신공학과 학사. 2007년 한양대학교 공과대학 전자컴퓨터통신공학부 석사. 현재 삼성전자 정보가전 소프트웨어팀. 관심분야는 운영체제, 파일시스템, 컴퓨터네트워크

박 재 석

2005년 한양대학교 공과대학 전자컴퓨터통신공학부 학사. 2007년 한양대학교 공과대학 전자컴퓨터통신공학부 석사. 현재 삼성전자 정보가전 소프트웨어팀. 관심분야는 운영체제, 파일시스템, 컴퓨터네트워크

이 주 훤

2005년 인하대학교 항공우주공학과 학사. 2005년~현재 한양대학교 공과대학 전자컴퓨터통신공학부 석사 재학중. 관심분야는 운영체제, 파일시스템