7

저장장치 결함과 소프트웨어의 설계

한양대학교 ▌원유집* · 김영진

1. 서 론

우리는 21세기 정보의 홍수시대에 살고 있다. 급속한 정보의 팽창은 크게 3가지 축에 근간을 두고 있다. 자료의 저장을 담당하는 저장장치, 자료의 교환을 담당하는 네트웍 미디어, 그리고 최종사용자 단계에서 자료의 효율적 생성과 소모를 담당하는 단말기가 그것이다. 물론, 이 기술들의 기저에 컴퓨터 CPU 기술의 발달, 자료의 압축/재생 알고리즘의 정교화 등 컴퓨터 기술 전반에 걸친 발전이 전제되어있음은 재론의 여지가 없겠다. 현대의 저장장치는 지난 20년간 무어의 법칙을 능가하는 급속한 저장장치 용량의 증가를 보여왔다. 이러한 저장장치 자료 집적도의 증가는 전자적인 부분과 기계적인 부분으로 구성되어 있는 하드디스크, 그리고 반도체 기술에 근거한 플래쉬 메모리에서 모두 나타나고 있다. 네트웍 속도의 급신장, Wibro, WiPAN, HSDPA 등 다양한 모바일 혹은 고정 무선 네트웍의 등장으로, 이제는 사용자들이 다양한 자료를 시간과 공간의 제약 없이 전송, 교환할 수 있게 되었다. 최근 들어 급속한 대중화되고 있는 다양한 콘텐츠 생산 기기(예: 디지털 카메라, 디지털 캠코더, 핸드폰 등)는 동영상, 사진 등 고품질, 고화질 멀티미디어 자료의 생성을 매우 용이하게 하였다. 캠코더, 디지털카메라 등으로 대변되는 콘텐츠 생성기기만으로는 자료 분량의 급속한 증대가 이루어질 수 없다. 콘텐츠의 "소모" 혹은 "소비"가 병행되어야 함이 필수이다. 캠코더, 디지털카메라 등으로 대비되는 컨텐츠 저작기기의 급속한 확산과 병행하여, 콘텐츠의 소비를 담당하는 고품질, 고사양의 모바일 멀티미디어 재생기기(예: Portable Multimedia Player(PMP), MP3 재생기, 핸드폰 등) 역시 급속히 대중화되고 있는 것이 현실이다. 저장장치, 네트웍, 그리고 이들의 생성과 소비를 담당하는 단말기 기술의 급속한 신장에 기인하여, 이제 개인이 다루어야 할 자료의 양이 급속히 증가하고 있으며, 곧 수 테라바이트를 넘어설 것으로 예측되고 있다.

본 고의 주제는 저장장치의 결함과 현대 소프트웨어이다. 어떠한 컴퓨터 부품도 결함에서 자유로울 수 없다. CPU, 메모리, 그래픽 카드, 하드 디스크 등 모든 컴퓨터 부품들은 결함에 노출되어 있다. 저장장치의 결함은 여타 부품의 결함과는 달리 하드웨어 손상이 자료의 손실로 연결된다. 하드웨어의 손상은 교체 등의 방법을 통하여 복구될 수 있으나, 자료의 손실은 경우에 따라서는 중대한 경제적 손실로 이어질 수도 있다. 하드웨어 교체비용을 훨씬 상회하는 손실이 야기될 수 있는 것이다. 경제적 손실은 없다 할지라도, 개개인이 생성한 자료들, 즉 자신이 직접 제작한 사진/동영상, 원고 등은 재 구매가 불가능하며, 매우 중요한 감성적 가치를 가지므로, 이들 자료의 손실 역시 사용자가 감내하기 어려운 사항들이다.

저장장치의 결함은 원천적으로 하드웨어 영역의 문제이다. 이제까지 대부분의 소프트웨어는 하드웨어와 소프트웨어 영역을 명확히 구분하여 하드웨어의 결함 가능성에는 큰 무게를 두지 않고 작성되었다. 곰곰이 생각해보면 당연한 접근 방식이다. CPU나 메모리에 문제가 생기면 해당 하드웨어를 교체해야지 소프트웨어가 어떻게 그 상황을 극복할 수 있겠는가. 여기에 약간의 차이점이 발생한다. CPU나 메모리는 결함이 발생하면 증상이 뚜렷이 나타나고, 대부분의 경우 결함이 발생하면 작동을 하지 않는다. 그러나, 저장장치는 CPU나 메모리와는 달리 상황이 좀 애매하다. 저장장치의 경우에는 결함이 일시적이거나, 아니면 결함이 발생한 영역이 전체 저장장치 용량의 극히 일부분에 해당할 수가 있는 것이다. 더구나 결함이 존재해도 저장장치는 외부(호스트)에서 볼 때 지극히 정상적으로 작동한다. 따라서, 하드웨어적 결함임에도 불구하고 소프트웨어적으로 처리할 수 있는 가능성과 여지가 존재하는 것이다. 현재 파일 시스템들은 저장장치의 이러한 기기적인 특성을 적절히 반영하지 않으며, "저장장치는 이

* 종신회원

상이 생기면 멈춘다(fail-stop)."는 매우 단순한 가정을 기저에 두고 설계되었다. 최근 들어 저장장치가 컴퓨터뿐 아니라 다양한 정보가전이나 모바일 기기 등에 사용됨에 따라 저장장치의 결함에 관련된 연구가 소프트웨어 분야에서도 매우 중요하게 강조되고 있다. 저장장치의 결함의 종류, 각 저장장치 결함에 대한 현대 파일 시스템의 대응방법, 저장장치 결함으로 인한 자료손실을 예방하기 위한 새로운 파일 시스템 구조 등이 그 예이다. 본 고에서는 저장장치의 하드웨어적 결함과 이의 소프트웨어적 대응의 문제점, 그리고 대응책 등을 소개하는 기회를 갖고자 한다.

2. 결함, 신뢰성, 일관성

2.1 결함과 오류

오류(error)와 결함(fault)을 먼저 명확히 정의하도록 하자. "오류"는 동작의 결과로 판단한다. "결함"은 "상태"에 관한 기술이다. 저장장치가 제대로 작동하지 못하는 상태에 있을 경우 이를 결함이라고 한다. 저장장치의 일부분에 결함이 존재하더라도, 해당부분을 다루는 입출력 명령어가 실행되지 않는다면 오류는 발생하지 않는다. 하드 디스크의 일부분이 베드섹터로 인하여 자료가 손실되었다 할지라도, 이 부분에 읽기 시도가 있기 전까지는 오류가 발생하지 않는다.

입출력 오류는 소프트웨어나 하드웨어 결함에서 연유한다[1,2]. 소프트웨어 결함의 원인으로는 디바이스 드라이버, 파일시스템의 버그 등이 있다. 대부분의 현대 파일 시스템은 입출력 성능향상을 위하여 비동기 쓰기(asynchronous write)기법을 사용하고 있다. 정전이나 운영체제의 오류 등으로 시스템이 비정상적으로 종료될 경우, 버퍼캐쉬에 저장된 내용과 저장장치에 저장된 내용이 일치하지 않을 가능성이 있다. 이런 현상을 파일 시스템이 오염되었다(file system corruption)고 하며, 파일 시스템의 중요 데이터 혹은 중요 부분이 손실되거나 사용이 불가능해 질 수 있다. 이러한 현상 역시 소프트웨어적 결함으로 간주하기로 한다.

표 1 하드 디스크의 결함 원인

순번	원 인
1	베드 섹터
2	디스크의 충돌과 긁힘
3	디스크를 구성하는 스핀들 모터[3] 손상
4	디스크를 떨어트리는 사용자 실수
5	전원장치로부터의 전기적인 문제
6	물리적인 접촉장치 문제로 인한 데이터 전달지연

저장장치의 결함은 결함의 위치에 따라 펌웨어와 하드웨어 2가지로 나눌 수 있다. 저장장치에 탑재되는 펌웨어가 점차 더 복잡해짐에 따라(현재 소스코드의 양이 40만라인 이상이다[4]), 펌웨어 코드에 결함이 발생할 확률이 점차 높아지고 있다[5]. 펌웨어의 결함은 자료를 잘못된 위치에 기록하거나(misdirected write), 기록에 실패했음에도 성공 메시지를 리턴하는 등(phantom write)의 오류를 발생시킨다. 하드웨어 결함으로는 구체적으로 다음과 같은 것들이 있다. 첫 번째 원인은 가장 흔히 발생하고, 경미한 경우로써 방치할 경우 한 순간에 모든 데이터가 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[3]와 디스크 사이의 간격에 끼어드는 오염 물질에 의해 발생되는 것으로써 데이터의 복구가 가장 어려운 오류이다. 세 번째 원인은 하드디스크를 구성하는 스핀들 모터[3] 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 사이에 인터페이스와 같은 물리적인 연결장치의 문제로 데이터 전달이 지연되거나 손실이 발생하는 경우이다. 필자의 경험에 의하면 의외로 저장장치 케이블의 결함으로 입출력 오류가 발생하는 경우가 많다.

2.2 신뢰성

신뢰성(reliability)는 저장 장치의 "물리적" 결함 가능성을 나타낸다. 파일 시스템의 일관성(consistency)이란 파일 시스템에 저장된 다양한 내용들이 "논리적"으로 상호 일치하는 가를 나타낸다. 시스템의 다양한 결함 가능성에 대비하여 컴퓨터를 설계하고자 하는 것은 매우 고전적인 주제이다[6]. 컴퓨터 구조, 메모리 설계, 저장장치 설계함에 있어서 추가적인 하드웨어 재원을 투입하고 이를 상호 연결하여 컴퓨터의 신뢰성 향상을 도모하는 것이다. 저장장치의 경우에는 디스크 미러링(RAID 1), 패리티 디스크(RAID 5)등을 두

어 결함에 대응한다. 이러한 기법들은 기업, 연구소등 주로 대용량 컴퓨팅 환경에서 사용되고 있으며, 개인용 컴퓨터나 모바일 저장장치까지 적용되기에는 아직 많은 무리가 있는 실정이다. 개인용 컴퓨터에서는 신뢰성보다는 가격이 최우선시된다. 따라서, 추가의 하드디스크 재원을 필요로 하는 디스크 미러링이나 패리티 디스크 등의 접근방식은 아직까지는 현실성이 없다. 현재 급속히 확장되고 있는 정보가전에서의 상황은 오히려 이보다 열악하다. 정보가전들은 근본적으로 기존의 컴퓨터와는 다른 요구사항을 지니고 있다. 정보가전은 기존 가전제품의 신뢰성과 가격수준을 유지해야 한다. 동시에 멀티미디어 정보가전은 기존의 범용 컴퓨터와 비슷한 수준의 연산, 저장능력을 제공해야 한다. 정보가전에서 저장장치의 신뢰성을 향상하는 문제는 더욱 어렵다 하겠다.

2.3 저장장치 결함의 빈도

이제까지 저장장치 결함의 종류에 대하여 기술하였다. 이제 가장 중요한 문제가 남았다. 결함의 빈도이다. 결함의 종류가 매우 다양하고, 그로 인하여 발생하는 오류가 치명적이라 할지라도, 결함 확률이 10-20라고 한다면[1] 저장장치의 결함은 중요한 논제가 아닐 것이다. 그렇다면, 하드 디스크가 얼마나 자주 망가질까? NAND 플래쉬 기반의 USB 메모리가 얼마나 자주 결함이 발생할까? 저장장치의 결함과 관련한 각종 통계자료들은 매우 민감 한 내용이기 때문에, 제조사들은 해당 내용을 거의 공개하지 않는다. 대부분의 컴퓨터 사용자들은 저장장치가 결함이 생길 가능성을 그리 크게 생각하지 않고 있다. 최근 들어 저장장치 결함과 이의 소프트웨어적 대응에 대한 관심이 높아짐에 따라, 저장장치 오류의 빈도, 종류, 특성 등을 분석한 연구결과가 학계에서 많이 발표되고 있다. 최근 발표된 연구결과에 의하면, 데이터 용량의 급속한 증대와 대용량 저장장치의 확산으로 인하여, 하드 디스크가 결함이 매우 쉽게 발견되고 있으며[7], 저장장치의 결함은 제품 사양서에 명시되어 있는 것보다 훨씬 자주 일어난다고 한다[8]. 대부분의 하드 디스크들은 MTTF(Mean-time-to Failure)를 1,000,000에서 1,500,000시간으로 표시하고 있으며, 이는 연평균 0.88%의 결함률에 해당한다. 그러나, 하드디스크의 연평균 결함률은 2-4%에 달하고, 저가의 IDE 디스크가 고가의 SCSI 기반이나 FC-AL 하드디스크보다 결함률이 그리 나쁘지 않은 것으로 보고되었다[8]. 저장장치의 결함은 생각보다 우리에게 매우 가까이 존재하고 있는 것이다.

2.4 일관성

저장장치의 일관성(consistency, integrity)이란 파일 시스템의 메타 데이터가 나타내고 있는 파일 시스템의 상태와 실제 파일 시스템의 상태의 일치하는지 여부, 데이터가 "옳게" 저장되어 있는지의 여부를 일컫는다. 일관성과 관련한 간단한 예를 들어 보도록 하겠다. 파일 시스템의 비트맵에서는 특정 데이터 블록이 사용 중으로 표시되어 있는데, 실제로 해당 데이터 블록은 어떤 파일에서도 사용하고 있지 않는 경우가 일관성이 깨진 경우이다. 이와 반대로, 더욱 심각한 상황은 실제 파일에 할당이 되었음에도 불구하고 해당 블록이 미사용으로 표시되어 있는 경우이다. 이 경우는 해당 데이터 블록이 다른 파일에 할당되어 하나의 블록을 두 개의 파일이 공유하는 현상이 발생할 수 있다. 바이러스나 침투공격에 의해서 특정 파일의 내용이 비 정상적인 경로로 변형되는 것도 일관성이 깨진 경우이다.

파일 시스템의 일관성은 저장장치의 물리적 결함이나 소프트웨어 버그 등, 악의적인 공격에 의해 손상될 수 있다. 자료의 일관성(file system consistency)를 유지하는 다양한 기법들이 제공되었으며, Sivathanu 외[2]에서는 일관성 유지기법들을 두 가지 분류방식에 따라 분류하고 있다. 메커니즘의 특성과 메커니즘이 구현된 계층에 따른 분류이다. 먼저 대응기법은 일관성 유지를 위한 특성에 따라 분류할 수 있다. 첫째, 파일 시스템이 일관성 보장을 위한 근본 메커니즘을 제공하는 기법들이다. 저널링[9-12], 읽기전용 파일 시스템[13], 트랜잭션 기반 파일 시스템[14,15]이 이에 해당한다. 둘째, 일관성에 문제가 생겼을 경우 이를 발견하는 접근방식들이다. 이들 기법들에서는 데이터 블록별로 혹은 파일 단위로 체크섬이나 데이터 미러링, CRC(Cyclic Redundancy Check), 패리티 등의 추가정보를 삽입하여, 입출력시에 해당 자료들의 일관성이 유지되고 있는지를 검증한다. 세 번째로, 일관성의 문제가 생겼을 경우 이를 발견하고 수정하는 기법들이다. 수정을 위해서는 두 번째 언급했던 기법들에 추가로 ECC(Error Correction Code), FEC(Forward Error Correction)등을 위한 정보를 추가로 삽입한다.

이러한 일관성 유지기법들은 해당 기법에 실제 구현되는 계층에 따라 구분될 수 있다. (i) 하드웨어 계층, (ii) 디바이스 펌웨어 계층, (iii) 파일 시스템 계층, 그리고 (iv) 응용 프로그램 계층이 그것이다. 하드디스

1) SHA-1 해쉬함수의 충돌확률

크나 플래쉬 등에서 사용하는 ECC 기법이 하드웨어 계층에서 사용되는 일관성 유지 방법에 해당한다. 이 외에 하드웨어 레이드 등도 있을 수 있다. 소프트웨어 레이드나 NAS(Network Attached Storage)등은 소프트웨어 수준에서 패리티나 inode 등에 대한 일관성 검사를 수행하고 있는데, 이는 위에 언급한 두 번째의 경우, 즉 소프트웨어에서 일관성을 유지하는 기법에 해당한다. 이 두 가지 기법들은 대부분의 저장장치에서 채택하고 있다. FSCK 나 저널링 기반의 파일 시스템이 일관성 유지를 보다 효과적으로 하는 기법이다. 최근, 저장장치 오류에 대한 중요성이 강조되면서, 자료의 일관성 점검을 위하여 추가 정보를 관리하는 실험적인 파일 시스템들이 제안되었다[14-17]. 솔라리스의 zfs는 64비트 체크섬을 모든 데이터 블록에 적용하여 데이터 블록이 손상되었는지 여부를 정기적으로 검사한다[16]. PFS는 모든 데이터 블록에 해쉬값을 저장하여 자료 입출력시에 데이터 블록으로부터 계산된 해쉬값과 저장되어있는 해쉬값을 비교하여 자료의 손상 여부를 판단한다[17]. 응용 프로그램수준에서의 일관성 유지 기법들은 주로 해킹 등의 악의적인 접근으로부터 파일 시스템 데이터를 보호하는 것에 중점을 두고 있다. 그 예로 TRIPWIRE[18], I3FS[19], SAMHAIN [20], Osiris[21] 등이 있다. 이들은 파일의 체크섬(TRIPWIRE), 파일의 PGP 키값(SAMHAIN)등을 저장하여 파일의 일관성을 점검하고 비정상적인 파일 변경을 발견한다.

3. 저장 장치결함과 파일시스템

3.1 파일 시스템 대응 기법의 분류

이제 본격적으로 저장장치의 물리적 결함이 있는 경우 소프트웨어가 어떻게 작동하는지를 알아보기로 하자. 우리는 두 가지 방법을 사용하여 하드디스크의 오류를 생성한다. 첫 번째는 하드 디스크의 특정섹터를 물리적으로 손상시켜서 베드섹터를 만들어 낸다. 두 번째는, 가상 스토리지 계층을 개발해서 특정 주소의 블록에 입출력이 들어오면 오류를 리턴한다. 가상 스토리지 계층은 디바이스 드라이버와 실제 디바이스 사이에 존재한다. 첫 번째 방법은 베드섹터의 읽기 오류실험에, 두 번째 방법은 베드섹터의 쓰기 오류실험에 사용되었다. 본 실험에 필요한 모든 소프트웨어는 자체적으로 개발되었다[22].

저장장치의 물리적 결함은 대부분의 경우 잠재해 있다. 즉, 결함이 실제 존재할 지라도, 결함이 존재하는 부분을 사용하기 전까지는 그의 존재 여부를 알 수가 없다. 하드 디스크의 경우 읽기 명령이 실패하면, 컨트롤러에서 정의한 횟수만큼 읽기를 재시도 한다. 일반적인 경우 이런 재시도의 횟수는 60-100회 정도되며(물론 제품마다 다르다), 결과적으로 성능의 치명적인 저하를 가져오게 된다. 이를 방지하기 위하여, 주기적으로 파일 시스템을 스캔하여, 결함이 있는 섹터들을 추출해내는 (disk scrubbing) 이라는 기법을 사용하기도 한다[23].

우리는 하드디스크에 임의로 베드섹터를 발생시켜서 베드섹터가 발생된 부분을 읽을 때, 실제로 파일 시스템이 어떻게 반응하는지에 대한 실험을 진행하였다. 이러한 분류방법은 IRON 파일 시스템에서[4]에서 제안되었다. 크게 두 가지 측면에서 파일 시스템의 대응을 분석하였다. 첫 번째는 저장장치의 결함으로 인하여 입출력에 오류가 발생했을 경우, 오류코드를 어느 정도 자세하게 분석하느냐 대한 평가이다. 베드섹터 존재로 인하여 작업이 실패했을 때, 파일 시스템이 제대로 베드섹터로 인한 오류의 존재를 제대로 발견하는 것이 주안점이다. 추후 구체적으로 기술하겠지만 놀랍게도 베드섹터로 인하여 입출력이 실패했음에도 불구하고, 파일 시스템은 계속적으로 정상 실행하는 경우가 많았다. 두 번째는, 오류가 발생했을 때 파일 시스템이 어느 정도의 복구동작을 수행하는가에 대한 평가이다. 저장장치에 결함이 존재하여 입출력 오류가 생겼을 경우, 파일 시스템은 이를 무시할 수도 있고, 이를 치유하기 위한 특정한 액션을 취할 수도 있다. 우리는 이것을 몇 가지 단계로 나누어 파일 시스템이 채택하고 있는 방식을 조사한다.

표 2는 파일 시스템의 오류검출, 오류 해결에 관련된 대응 수준을 도식적으로 표현하고 있다. 먼저 어느 정도 자세히까지 저장장치의 결함으로 인한 입출력 오류를 분류하는 지를 보기로 한다. 우리는 오류 발견에 대해서 4가지의 수준을 정의한다[24,25]. 저장장치에서 오류가 발생하더라도 파일 시스템에 전달되지 않는 경우(무시), 파일 시스템에 에러코드가 전달되는 경우(에러코드), 오류가 발생되었을 때 특정 자료구조를 검사하는 경우(특정자료구조 검사), 그리고, 에러 검출코드를 이용하여 에러를 발견하는 경우이다. 파일 시스템에서 행해지는 오류 복구 수준을 여섯 가지로 분류한다. 먼저 저장장치에서 오류가 발생하더라도 아무런 해결책이 실행되지 않는 경우이다. 이를 "무시"라고 표현한다. 두 번째, 저장장치에서 오류가 날 경우 사용자에게 상황을 에러코드를 통하여 전달한다. 이를 우리는 에러전달이라고 한다. 에러가 전달될 경

우 모든 판단은 응용 프로그램에게로 유보되며, 운영체제나 파일 시스템은 다른 액션을 취하지 않는다. 세 번째, 오류가 발생하면 파일 시스템을 정지시키거나 읽기 파일 시스템으로 다시 마운트한다. 네 번째로, 입출력 오류발생시 해당 작업을 다시 시도하는 것이다(재시도). 재시도의 주체가 어디냐 하는 것도 중요한 평가요인이 된다. 파일 시스템의 읽기가 실패했을 경우, 파일 시스템은 fsck를 통해서 파일 시스템의 무결성을 점검할 수 있다. 실제로 작동중인 시스템에서 운영체제가 자의적으로 fsck를 실행하는 것이 옳은 것인가에 대한 논의는 본 고의 범위를 벗어나므로 배제하도록 하겠다. 마지막으로 파일 시스템의 입출력이 실패했을 경우, 복제된 자료를 이용하여 파일 시스템을 복구하는 것이다. 이 경우는 손상된 자료들에 대한 복사본의 존재가 미리 전제되어야 한다.

표 2에 기술된 내용들을 좀 더 자세히 살펴보도록 한다. 먼저 오류 검사의 구체성이다. 오류를 전혀 인지하지 못하고, 물리적인 오류에 대해 어떠한 대처도 하지 않을 경우, 해당 파일 시스템은 물리적 오류에 취약하게 된다. 디바이스 드라이버 영역[27]에서 에러 코드를 전달 받아 합당한 처리를 진행하거나[2] 또는 특정 자료구조의 검사를 통해 오류를 탐지할 경우에는 물리적인 오류에 대한 강인성이 높다 할 수 있다. 다음으로 복구의 구체성을 보도록 하자. 물리적인 오류가 발견되었음에도 불구하고 어떠한 복구 과정을

표 2 파일 시스템에서 수행하는 오류에 대한 검출 및 복구 수준정의

구분	수준 정의	정의된 수준 설명
오류 검출	무시	저장장치에서 오류가 발생하더라도 파일시스템은 알지 못함
	에러코드	디바이스 드라이버 영역에서 파일 시스템으로 에러 코드를 전달
	특정 자료구조 검사	파일 시스템 슈퍼블록 검사와 같은 방식으로 특정 자료구조 검사
	검사[2]	에러 검출 코드를 이용한 발견
오류 복구	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[26]와 같이 논리적으로 파일시스템을 복구
	복제 기법[2]	특정 영역을 미리 복제하고, 오류가 발생했을 때 복제본으로 복구

수행하지 않는 경우, 물리적인 오류에 대한 복구가 매우 미흡하다고 할 수 있다. 반대로, 미리 일정 영역을 복제하고 오류가 발생했을 때 복제본을 통해 복구하는 경우를 복구 측면에서 매우 강인하다고 표현할 수 있겠다.

3.2 저장장치 결함 발생

우리는 리눅스 계열에서 가장 널리 사용되고 있는 ext3 파일 시스템이 저장장치의 물리적 결함에 대해 어떻게 대응하는 지를 실제 실험을 통하여 관찰하였다. 자체 제작된 디스크 결함 발생기는 하드 디스크의 섹터 번호를 입력 받아 해당 섹터의 읽기가 불가능하도록 섹터의 내용을 오염 시킨다[22]. 해당 섹터를 읽고자 할 경우 하드 디스크 컨트롤러는 이를 베드섹터로 인식하게 된다.

파일 시스템에는 매우 다양한 형태의 자료들이 존재하며, EXT3 파일 시스템 역시 예외는 아니다. 파일 시스템에 존재하는 자료 중 가장 많은 부분을 차지하는 것이 파일의 데이터 들이다. 그리고, 파일의 관리정보를 가지고 있는 inode 블록, 파일 시스템의 빈 공간 관련 정보를 나타내는 각종 비트맵들, 파일 시스템 전체의 관리정보를 가지고 있는 슈퍼 블록등이 존재한다. 이 외에, 파일 시스템의 일관성 유지를 위한 정보를 가지고 있는 저널파일이 있다. 저널파일의 데이터 블록들은 ext3 저널 방식의 구조적 특성상 저널의 슈퍼블럭, 저널의 디스크립터 블록, 커밋블럭 등으로 구성되어 있다[10]. 파일 시스템입장에서 위에서 언급된 다양한 블록들은 서로 다른 의미를 가지고 있으며, 중요성 역시 다르다. 본 실험에서는 ext3 파일 시스템에 존재하는 블록들을 표 3에서와 같이 분류한다. 각 블록에 해당하는 내용이 저장장치의 결함으로 인하여 손실될 경우 파일 시스템이 어떻게 행동하는지를 관측하도록 한다.

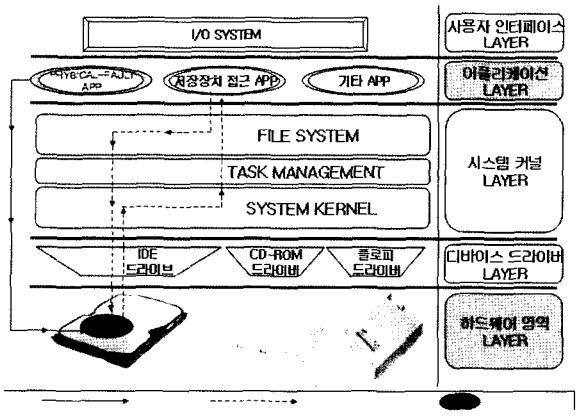

그림 1 에러 발생 구조

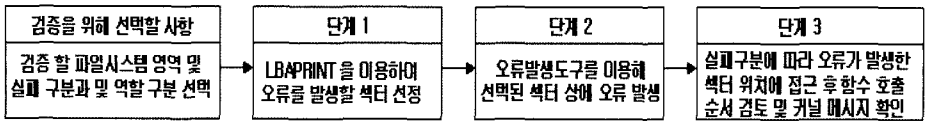

그림 2 파일 시스템 검증 순서

3.3 테스트를 위한 결함 발생과 부하 생성

우리는 파일 시스템에서 가장 자주 사용되는 9개의 시스템 콜을 선택하여 파일 시스템에 적용하였다. create, read, write, link, unlink, mkdir, rmdir, mount, umount 들이 그것이다(표 5-8). 각 시스템 콜들은 각자 자신들이 필요로 하는 파일 시스템의 각종 블록들에 대해서 입출력을 수행한다. 파일을 생성하는 시스템 콜인 create의 경우, 부모 디렉토리의 inode, 부모 디렉토리의 데이터 블록, 새로이 생성된 파일의 inode, 그리고 파일 시스템의 inode 비트맵을 저장하고 있는 블록들이 사용된다. 우리는 각 시스템 콜의 코드를 분석하여 그들이 사용하는 파일 시스템의 블록들을 먼저 추출하였다. 그리고, 각 블록들에게 입출력 오류를 생성하여 해당 파일 시스템이 어떻게 대응하는지를 분석하였다.

그림 2는 본 연구에서 채택한 접근방법을 도식적으로 표시하고 있다.

물리적인 오류(읽기와 쓰기)를 발생시킨 후, 파일 시스템이 정확하게 오류를 발견하고 복구를 수행하는지의 여부를 검사한다. 파일 시스템이 택하고 있는 오류 발견 기법, 오류 복구 기법들을 구체적으로 알아내는 것은 매우 복잡하고 정교한 작업이다. 파일 시스템의 각 영역에 입출력 오류를 발생시킨 후, 부하를 가하고, 커널 메시지 등을 통해서 파일 시스템의 대응을 분석한다.

표 3 파일시스템 블록의 종류

기호	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 4 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
γ	현재의 에러를 사용자에게 알려주는 복구방법
δ	β와 γ의 복구동작을 동시에 수행하는 방법
ε	γ와 현재의 작업을 반복하는 복구 방법

3.4 저장장치 결함에 대한 파일 시스템의 대응

표 5 읽기실패의 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 6 읽기실패의 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		γ	ε							
Write			β	γ	α							
Link		δ	δ									
Unlink	α	δ	α	γ	α							
Mkdir	δ	δ	δ			δ						
Rmdir	α	δ	α	γ								
Mount		δ				δ	δ	δ				
Umount						δ		δ				

표 7 쓰기실패의 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 8 쓰기실패의 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					

표 5 - 8은 읽기 오류와 쓰기 오류 시에 ext3 파일 시스템이 수행하는 오류 발견 전략과 오류 복구전략을 도식적으로 나타내고 있다. 표의 셀 내에 어떠한 기호도 적혀있는 않은 것은 시스템 콜이 해당 파일 시스템 영역을 접근하지 않는 것을 의미한다. 표의 가로축은 파일시스템의 각 영역을 나타내고 있으며, 세로축은 적용된 부하를 나타낸다. 표의 각 셀의 부호는 표 4에서 정의된 것과 같이 파일 시스템이 택하고 있는 구체적인 전략을 표시하고 있다. 표 7과 8을 예로 들어 설명하도록 하겠다. 이 표는 파일 시스템이 쓰기에 실패했을 경우 어떤 행동을 하는가를 요약하고 있다. Create를 보도록 하자. Create는 a(inode bitmap), b(inode table), c(data bitmap)등을 사용하는데 실제로 해당 블록들에 쓰기를 실패했을 경우 α의 전략을 택하는 것으로 분석되었다. α는 아무런 대응도 하지 않는 것을 의미한다. 결론적으로 파일 생성시에 inode bitmap, data bitmap을 갱신하는 것에 실패 했다 할지라도, EXT3 파일 시스템은 어떠한 대응도 없이, 사용자에게 성공플래그를 리턴한다.

우리는 복구표를 통해서는 매우 다양한 복구수준이 적용되고 있음을 한눈에 알 수 있었다. 테스트된 모든 시스템 콜들은 "읽기" 실패 시에 에러코드를 발생시키고 있다. 복구에 있어서는 읽기 실패가 발생한 블록의 종류와 시스템 콜의 종류에 따라 매우 다른 복구 방식을 따르고 있다. read 시스템 콜인 경우Inode table, Data block, Data indirect(파일의 포인터 블록) 영역을 각각 읽는다. Inode table 읽는 것을 실패하면 현재의 작업을 중단하고, 현재의 오류 상태를 사용자에게 알려준다. Data block 읽기를 실패하면 현재의 오류 상태를 사용자에게 전달하며 그 외의 동작은 취하지 않는다. Data indirect블록 읽기를 실패할 경우 다른 접근 방식을 취하고 있다. 유일하게 현재의 오류 상태를 사용자에게 알려주며 재 접근을 시도하고 있다. 그래서 δ, γ, 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 사용 하고 있다.

다음은 쓰기 실패에 대한 EXT3 파일 시스템의 대응을 분석해 보도록 한다. Ext3 파일 시스템은 쓰기 실패에 대해서 대부분의 경우 어떠한 대응도 하고 있지 않다는 놀라운 사실을 발견하였다. 저널 파일의 슈퍼블록에 쓰기를 실패하였을 경우에도 어떠한 대응도 하고 있지 않다. 예외적으로 저널파일에서 슈퍼블록을 제외한 다른 부분에 대해서 쓰기가 실패한 경우에는 현재 작업을 중단하고 오류를 사용자에게 전달하는 기법을 채택하고 있다.

3.5 결함(defect)인가 아니면 사양(feature)인가?

이번 테스트를 통해서 몇 가지 재미있고 매우 중요한 사실을 발견하였다. 저장장치의 물리적 결함에 대해서 파일 시스템이 적합한 대응을 하고 있지 못하다는 사실이다. EXT3 파일시스템은 베드섹터 발생으로 인하여 읽기에 실패하더라도 어떠한 복구 동작도 하지 않거나 사용자에게 오류에 대한 정보만을 전달한다. 복구 동작 또한 논란의 여지가 많다. 저널 영역에서 읽기에 실패하였을 경우에만, 현재의 작업을 중단하는 복구 노력이 있을 뿐 대부분의 영역에서는 어떠한 복구 동작도 하지 않는다. 매우 특이한 사항은, 저널 슈퍼블록에 대해서는 저널의 다른 블록들과는 달리 읽기나 쓰기의 실패에 대해서 어떠한 점검이나 복구도 시도하지 않고, 실패가 그냥 무시된다는 것이다. 이는 실제로 매우 치명적인 결과를 가져올 수 있다. 저널 슈퍼블록에 적절한 자료를 기록하는 데 실패한 후에 파일 시스템이 정전으로 인하여 크래쉬 되었다고 가정하자. 시스템 재 구동 시에, 파일 시스템은 저널 파일에 기록된 내용에 근간하여 파일 시스템 복구를 시도할 것이다. 만약, 저널 슈퍼블록이 잘못된 내용을 가지고 있는 경우, 파일 시스템은 잘못된 내용에 근간하여 복구를 시도할 것이며 따라서 파일 시스템 전체의 일관성이 깨어지고 자료가 손실되는 상황이 발생할 수 있게 된다. 이는 발생할 수 있는 다양한 위험 시나리오에 극히 일부분에 해당한다.

그렇다면, 이제까지 기술된 EXT3 대응 수준은 과연 결함일까 아니면 파일 시스템의 사양일까? 이는 하드웨어의 물리적인 결함에 대해 소프트웨어가 어느 정도까지 책임을 져야 하는가 대한 소프트웨어의 역할 정의 문제로 귀추 된다. 일 예를 들어보자. 응용 프

로그램의 write()가 디바이스의 물리적 결함으로 실패하였다. 이 경우 파일 시스템은 어떤 행동을 취하는 것이 옳은가? 간단하게 세 가지의 대응책이 있을 수 있다. 첫 번째는 실패를 무시하고 진행하는 것이다. 두 번째는 실패를 사용자에게 알리는 것이다. 세 번째는 재시도를 하는 것이다. 언뜻 생각하기에는 재시도를 하는 것이 가장 멋있어 보인다. 그러나, 이것은 파일 시스템의 근본 정의와 관련이 있기에 단순한 문제가 아니다. 세 번째 방식에 의하면, 응용 프로그램이 생성한 쓰기 명령에 대해서, 파일 시스템이 자의적으로 다수의 동일한 쓰기명령을 저장장치에 전달한다. 만약 응용 프로그램이 내부적으로 자체적인 스케쥴링 기법을 이용하여 쓰기 명령을 생성하는 상황이라면, 파일 시스템이 자의적인 대처가 오히려 응용프로그램의 실시간성을 해칠 수도 있는 것이다. 그러면, 첫 번째 방법이 과연 제일 안 좋은가? 그 부분에 대한 판단은 독자에게 넘기기로 한다.

"파일 시스템은 응용 프로그램의 입출력 요청을 적절히 변형하여 저장장치에 전달하는 무지한(dumb) 소프트웨어 계층이다"라고 정의한다면, 하드웨어의 결함에 대해서는 최소한의 대응만을 하는 것이 옳은 접근 방식일 것이다. 그러나, 최종사용자는 기기의 내부가 소프트웨어와 하드웨어로 구성되어 있다는 것에는 전혀 관심이 없다. 자료가 제대로 저장되었는지, 제대로 재생이 되는지가 지고의 관심사이다. 이런 측면에서 본다면, 파일 시스템은 저장장치의 물리적 결함을 최대한 복구할 수 있도록 설계되어야 하는 것이다. 최근 들어 저장장치나 파일 시스템에 충분한 유연성을 부여하여 탑재된 자료의 용도에 따라 파일 시스템의 대응 수준을 결정하는 기법이 제안되기도 하였다[28]. 파일 시스템 대응수준의 결정은 학문적 입장에서는 파일시스템의 역할에 관한 철학적 토론 주제이며, 실제 기업이나 개발자 입장에서는 책임소재에 관련된 매우 민감한 사안이라 할 수 있겠다.

4. 저장장치 결함과 멀티미디어 정보가전

멀티미디어 정보가전은 향후 컴퓨터 기술이 총체적으로 집적될 주요 플랫폼이다. 정보가전에는 CPU, 메모리, 저장장치(하드디스크, 플래쉬)등의 모든 컴퓨터 컴포넌트들이 집적되어 있다. 정보가전은 가전제품의 신뢰성을 만족시켜야 한다. 이는 기기가 매우 거친 환경에서도 무리 없이 작동해야 한다는 것을 의미한다. 예를 들어, 이삿짐처럼 운반하다가 떨어뜨릴 수도 있고, 아이들이 던진 장난감에 의해서 큰 충격을 받기도 하고, 또 40도 이상의 고온에서도 장시간 작동하기도 한다. 따라서, 저장장치의 결함 가능성이 더욱 커지는 것이다. 우리는 기존의 멀티미디어 재생 소프트웨어들이 저장장치에 결함이 존재할 경우 어떤 행태를 보이는가를 실험을 통해서 고찰하였다.

4.1 멀티미디어 파일과 저장장치의 결함

멀티미디어 서비스를 제공하는 대표적인 어플리케이션인 멀티미디어 플레이어와 MP3 플레이어를 들 수 있다. 멀티미디어 파일의 크기는 음악이나 사진을 담고 있는 수십 Kbyte 의 비교적 작은 파일에서부터 2시간 분량의 고화질 영상을 저장하는 수 Gbyte에 달하는 매우 큰 파일까지 매우 다양하다. 파일 내에서 오류가 발생한 위치에 따라 손실되는 정보의 양이 다를 수 있다. 예를 들어 포인터 블록에 오류가 생겼을 경우, 포인터 블록을 통해서 접근되는 모든 블록들은 손실된다. 데이터 블록에 오류가 생겼을 경우 해당 블록에 저장된 자료들만이 손실될 뿐이다. 본 실험에서는 멀티미디어 재생 소프트웨어가 저장장치 오류에 대해서 보이는 반응을 분석하였다.

먼저 표 9에서와 같이 읽기 오류 시에 프로그램의 반응을 6가지로 분류하였다. 반응 수준은 대문자 알파벳으로 구성되어 있는데 읽기 실패 시 가장 좋은 반응 순서대로 열거 하였다. 단순한 인터페이스 오류로 인한 실패일 수 있기 때문에 해당 부분을 재시도하는 상황과 읽기가 실패 된 부분을 점프하여 다음 프레임부터 재생하는 상황, 정상적으로 읽혀진 부분까지 재생 후 정지되는 상황, 읽기실패가 되었을 경우 프로그램이 자동으로 종료되는 상황, 파일의 끝에 도달한 것과 같은 상황, 마지막으로 가장 좋지 않은 프로그램 자체의 깨짐 상황을 보였다. 표 10은 읽기 시도가 실

표 9 읽기실패 시 화면 수준표

기호	상황
A	재시도
B	읽기실패 부분 점프
C	정지화면
D	정상종료와 같은 상황
E	프로그램 자동 종료
F	프로그램 크래쉬

표 10 에러 메시지 수준표

기호	상황
A	정확한 메시지 출력
B	부정확한 메시지 출력
C	메시지 출력 없음

패하였을 경우 정확한 오류 상황이 사용자에게 전달되는 지의 여부를 판단하는 기준을 정리하였다. 우리는 멀티미디어 파일의 임의 부분에 베드섹터를 발생시켰다. 오류가 있는 파일을 각각의 플레이어를 통해 재생시킨 후 읽기 실패가 이루어졌을 때 화면의 상황을 관찰하였으며, 정확한 에러 메시지를 출력하고 있는지를 확인하였다. 또한 멀티미디어 파일의 정보를 담고 있는 헤더 부분에 읽기실패가 발생하였을 경우 플레이어의 에러 메시지 상황을 살펴보았다.

4.2 멀티미디어 소프트웨어의 반응

우리는 동영상과 음악 컨텐츠의 재생에 대한 저장장치 오류의 효과를 분석하였다. 동영상 재생 소프트웨어로는 Avifile, MTV, Mplayer-0.90의 세가지를 테스트 하였으며, 음악 재생 소프트웨어로는 realplayer-7.0, Mplyer-0.90, freeamp의 세가지를 테스트 하였다.

표 11은 동영상 재생 소프트웨어의 반응을 정리하고 있다. Avifile과 MTV는 읽기에 실패한 경우, 이미 읽은 데이터들을 재생한 후 화면이 정지되었다. MPlayer-0.9 같은 경우 Avifile이나 MTV와는 달리 재시도를 수차례 진행 한 후, 읽기에 실패한 영역은 건너뛰고 계속 재생을 진행하였다. 동영상 자료의 특성상 한 블록이 손실되는 것은 전체적인 자료의 내용에 큰 영향을 미치지 않는다. Mplayer-0.9는 이러한 특성을 매우 효과적으로 사용하여 잘 설계되었다. 다음은 읽기 오류 시 발생하는 에러메시지의 종류이다. Avifile과 MTV는 에러 메시지를 확인 할 수 없었으며, 읽기를 모두 성공했음을 알리는 "Success" 메시지만을 보여주었다. 하지만 MPlayer-0.9는 "WARNING" 메시지와 함께, 에러가 발생한 정확한 위치와 "Input/output error"메시지를 알려주고 있어 매우 정교하며 물리적인 오류에 대해서도 강인함을 확인 할 수 있었다. 두 번째 실험은 멀티미디어 파일의 정보를 담고 있는 헤더정보 부분의 오류에 대한 멀티미디어 소프트웨어의 대응이다. Avifile은 "Sorry, this file format is not recognized supported. If this is an AVI, ASF or MPEG stream, please contact the auther!"이라는 메시지를 출력하고 실행을 정지하였다. 재생에 필수적인 정보가 손실되었으므로, 이에 적절한 대책을 취하고 있었다. MTV의 경우 경고 메시지는 있었으나 메시지 내용은 정확하지 않았다. MpegTV Alert 라는 창 제목이 내용은 이렇다. "Warning! Hum,,, This doesn't look like an MPEG-1system or Video stream. Use Xaudio to play MPEG Audio or MP3 streams." 이 내용은 MPEG-1 시스템이 아니거나 비디오 시스템이 아니라는 표현을 사용하고 있으나 이것은 부정확한 메시지다. 헤더 정보의 문제나 포맷의 문제를 지적한 것이 아니기 때문에 파일에 문제가 있음을 알 수 없기 때문이다. MPlayer-0.9는 정확한 이유까지는 알려주고 있지 않지만 파일을 읽는 모듈에서 파일을 열 수 없음을 나타내고 있다. 메시지 창을 확인해보면 다음과 같다. "〈exception〉 AsfReaderHandler: FATAL: Could not open asf stream. 〈reader〉: can't open file" 파일을 읽는 모듈에서 헤더 정보를 읽지 못하기 때문에 이와 같이 읽기 실패로 인한 파일을 열 수 없음을 알려주고 있다. 지금까지 멀티미디어 동영상 파일을 재생할 수 있는 프로그램이 동영상 파일의 베드섹터 즉 읽기 실패에 직면했을 경우 어떠한 행동을 보이고 있는지와 헤더 정보에 읽기 실패가 이루어졌을 경우 메시지를 살펴봄으로써 어플리케이션 영역에서 물리적인 오류에 대한 행동과 대처 수준을 살펴보았다. 이번 실험은 동영상 멀티미디어 플레이어와 같은 실험 방식으로 MP3 플레이어에 대한 실험을 진행하였다. 표 12는 실험 결과를 요약하고 있다.

MP3 파일은 하나의 블록에 저장되는 음악정보의 양이 상대적으로 크기 때문에 읽기 실패의 영향이 상대적으로 심각하다. 표 12는 MP3 재생 소프트웨어의 대응을 요약하고 있다. RealPlayer는 결함이 있는 섹터를 읽었을 때, 프로그램이 종료되었다. 저장장치의 물리적인 오류에 대응이 미약함을 알 수 있다. 발생되는 에러 메시지 또한 정확하지 않았다. 메시지는 "Bus error" 였으며, 헤더 파일의 읽기 실패가 발생되었을 경우 또한 같은 메시지를 보였다. 플레이 중에 읽기가 실패했을 경우에는 읽기실패가 이루어진 블록을 알려주고 재시도를 수행한 후 해당 영역을 건너뛰어 재생을 계속하는 것이 가장 이상적인 대응방법이다.

표 11 멀티미디어 플레이어의 읽기실패에 관한 결과표

멀티미디어 플레이어	화면 수준	에러 메시지 정확도	헤더 파일의 읽기 실패 시 에러메시지
Avifile	C	c	a
MTV	C	c	b
MPlayer-0.90	A/B	a	a

표 12 MP3 플레이어의 읽기실패에 관한 결과표

멀티미디어 플레이어	화면 수준	에러 메시지 정확도	헤더 파일의 읽기 실패 시 에러메시지
RealPlayer 7.0	E	b	b
MPlayer-0.90	D	c	a
Freeamp (ZINF)	D	c	b

하지만 프로그램이 종료되고 에러 메시지 또한 추상적인 표현을 사용하고 있어 물리적인 오류에 매우 약함을 확인할 수 있었다. 그리고 헤더 파일의 읽기 실패 역시 정확하게 포맷에 문제가 있음을 알려줘야 하지만 일관되게 "Bus error" 메시지만을 보여주고 있는 것으로 봐선 에러 처리 루틴이 매우 협소하게 되어 있음을 짐작할 수 있다. 두 번째와 세 번째 어플리케이션인 MPlayer와 Freeamp는 읽기 실패가 이루어지면 음악이 정상적으로 종료 된듯한 화면을 보여주고 있으며, 이때의 에러 메시지 또한 성공을 나타내고 있다. 이 현상으로 알 수 있는 사실은 읽기실패가 이루어졌을 경우 파일의 끝에 도달했다고 어플리케이션이 단정짓기 때문에 나타나는 현상이라고 생각한다. 헤더 파일의 읽기 실패가 되었을 경우도 두 어플리케이션은 비슷한 메시지를 보여주고 있다. MPlayer 경우 "Falling back on trying to parse playlist/mnt/hdd/i.mp3"라는 메시지를 표시하여 해당 파일의 정확한 경로와 해당 파일의 헤더 부분 분석 시 읽기 실패 되었음을 전달한다. Freeamp는 "Freeamp can not play this file/stream. This file/stream may be corrupted" 라는 메시지를 출력하여, 파일이 손상되었음을 알린다. 하지만 헤더 파일의 읽기 실패일 경우는 파일 형식에 문제가 있다는 메시지나 파일 문법적인 문제가 있음을 알려줘야 하지만 그런 내용을 발견하지 못하였기에 부정확한 메시지라고 표현하였다.

5. 맺음말

우리는 이제까지 파일 시스템, 응용 프로그램 등의 소프트웨어와 저장장치 결함의 상관관계를 실제 실험을 통해 살펴보았다. 현대의 파일 시스템은 저장장치의 물리적 결함에 대해서 적절한 대응을 하지 않고 있음을 알 수 있었다. 응용 프로그램 역시 저장장치의 물리적 결함에서 연유하는 입출력 오류에 대해서, 전혀 대응 없이 설계되었음을 경험하였다. 모든 파일 시스템과 모든 응용프로그램에 대해서 전반적인 검증을 실시하지는 못했으나, 검사에 대상이 되었던 EXT3와 범용적으로 사용되고 있는 소프트웨어들은 저장장치의 결함에 대하여 만족할 만한 수준의 대응을 하지 않고 있었다. 이러한 분석결과가 가지는 의미는 매우 크다. 저장장치의 용량이 급속히 증대되고, 기기가 점점 복잡해지고 있으며, 더 열악한 물리적 환경에서 사용될 것이다. 이와 더불어 기기에 대한 가격 경쟁은 점점 심해질 것이다. 이와는 반대로 저장장치가 다루는 자료는 경제적으로 감성적으로 점차 더 중요한 내용을 다루게 될 것이다. 저장장치는 점차 결함에 많이 노출이 되고, 저장되는 자료는 점차 중요해지는 상황에서, 소프트웨어는 입출력을 다루는데 있어서 어떤 접근방식을 택해야 하는가? 이에 가장 선행되어야 하는 것은 파일 시스템의 역할을 명확히 정의하는 것이다. 가장 이상적인 것은 저장장치에 존재하는 다양한 종류의 블록들에 대해서 각 블록의 중요성에 적합한 파일 시스템의 역할을 정의하고 이에 기반하여 보장장치를 제공하는 것이다. 슈퍼블록, 비트맵 등의 메타데이터에 대해서는 복제와 복구 등에 대한 장치를 마련하고, 데이터 블록 등에 대해서는 최소한의 대응만을 구현하는 것이다. 또 한가지는 파일 시스템의 역할을 정의할 때, 파일 시스템이 사용되는 환경의 특성을 최대한 고려하는 것이다. 동영상 재생을 다루는 멀티미디어 정보가전용 파일 시스템이라면 데이터 블록에서 발생하는 읽기오류는 매우 사소한 사안이며, 읽기 재시도는 오히려 전체적인 시스템의 성능만 악화시키는 원인이 될 수 있다. 저장장치는 현대 컴퓨터 시스템의 핵심 요소중의 하나이다. 이제까지 살펴본 바와 같이 저장장치 결함에 대해서, 소프트웨어들이 그다지 많은 중점을 두지 않고 개발되었다. 보다 신뢰성 있고 경제적인 정보기기의 개발을 위해서는 기기의 결함에 대해 적극적, 구체적으로 대응하는 소프트웨어 계층에 대한 연구개발이 진행되어야 하겠다.

참고문헌

[1] G. Sivathanu, C. P. Wright, and E. Zadok, "Ensuring Data Integrity in Storage: Techniques and Applications," in ACM StorageSS '05, Fairfax, Virginia, USA, 2005, pp. 26-36.

[2] S. Gopalan, P. W. Charles, and Z. Erez, "Ensuring data integrity in storage: techniques and applications," in Proceedings of the 2005 ACM workshop on Storage security and survivability, Fairfax, VA, USA, 2005, pp. 26-36.

[3] Y. A. G. Al-Regib, "An Unequal Error Protection Method for Packet Loss Resilient 3-D Mesh Transmission," IEEE, 2002.

[4] P. Vijayan, N. B. Lakshmi, A. Nitin, S. G. Haryadi, C. A.-D. Andrea, and H. A.-D. Remzi, "IRON file systems," in Proceedings of the twentieth ACM symposium on Operating systems principles(Brighton, United Kingdom, October 23-26, 2005), Brighton, United Kingdom, 2005, pp. 206-220.

[5] G. Weinberg, "The Solaris Dynamic File System," 2004.

[6] D. A. Rennels, "Fault-Tolerant Computing," in Encyclopedia of Computer Science, A. Ralston, E. Reilly, and D. Hemmendinger, Eds.: International Thomson Publishing, 1999.

[7] S. Ghemawat, H. Bobiof, and S.-T. Leung, "The Google file system," in 19th ACM Symposium on Operatging Systems Principles(SOSP '03), 2003.

[8] B. Schroeder and A. G. Garth, "Disk Failures in the real world: What does an MTTF of 1,000,000 hours mean to you?," in USENIX Conference on File and Storage Technologies(FAST '07), 2007.

[9] M. I. Seltzer, G. R. Ganger, M. K. McKusick, K. A. Smith, C. A. N. Soules, and C. A. Stein, "Journaling Versus Soft Updates: Asynchronous Meta-data Protection in File Systems," in the Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, California, USA 2000, pp. 71-84.

[10] K. Sovani, "Linux: The Journaling Block Device". vol. 2006, 2006.

[11] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and Evolution of Journaling File Systems," USENIX 2005 Annual Technical Conference, General Track, pp. 105-120, April 2005.

[12] G. R. Ganger, M. K. McKusick, C. A. N. Soules, and Y. N. Patt, "Soft Update: A solution to the Metadata Update Problem in File Systems," ACM Transactions on Computer Systems (TOCS), vol. 18, pp. 127-153, May 2000.

[13] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in USENIX conference on File and Storage Technologies, Montorey, CA, USA, 2002, pp. 89-101.

[14] B. Liskov, Rodrigo, and Rodrigues, "Transactional File Systems Can be Fast," in 11th workshop on ACM SIGOPS European workshop: beyond the PC, 2004.

[15] E. Gal and S. Toledo, "A transactional Flash file system for Microcontrollers," in USENIX 2005 Annual Technical Conference, General Track, Anaheim, CA, USA, 2005, pp. 89-104.

[16] "Soliar ZFS: The most Advanced File System in the planet," http://www.sun.com/software/solaris/ds/zfs.jsp.

[17] C. A. Stein, J. H. Howard, and M. I. Seltzer, "Unifying File System Protection" in 2002 USENIX Annual Technical Conference, 2001, pp. 79-90.

[18] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: a file system integrity checker," in 2nd ACM Conference on Computer and communications security, 1994, pp. 18-29.

[19] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, "I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System," in 18th USENIX Large Installation System Administration Conference, 2004.

[20] http://www.la-samhna.de/samhain/, "SAMHAIN."

[21] http://osiris.shmoo.com/, "OSIRIS."

[22] 김영진, 원유집, and 김락기, "ROS : Reliability Verification Tool of the Storage," in JCCI 2007(Joint Conference on Communications and Informations) 보광피닉스, 2007.

[23] Thomas J. E. Schwarz, Q. Xin, E. Miller, D. Long, A. Hospodor, and S. Ng, "Disk Scrubbing in Large Archival Storage Systems," in IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems(MASCOTS '04), Volendam, Netherlands, 2004.

[24] K. Hannu, S. Heikki, and L. Fabrizio, "Detection of Defective Media in Disks," in Defect and Fault Tolerance in VLSI Systems, 1993., The IEEE International Workshop on, Venice, Italy, 1993, pp. 49-55.

[25] K. Hannu, S. Heikki, and L. Fabrizio, "Detecting Latent Sector Faults in Modern SCSI Disks," in Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS '94., Proceedings of the Second International Workshop on, Durham, NC, USA, 1994, pp. 403-404.

[26] 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.

[27] R. J. L. J. Mo, V. Anantharam, J. Walrand, "Analysis and COmparison of TCP Reno and Vegas."

[28] Y. Won, H. Chang, J. Ryu, Y. Kim, and J. Shim, "Intelligent Storage: Cross Layer Optimization for Soft Real-time Workload," ACM Trans. on Storage, vol. 2, pp. 255-282, Aug 2006.

원 유 집

1990 서울대학교 자연과학대학 계산통계학과 학사
1992 서울대학교 자연과학대학 계산통계학과 석사
1997 University of Minnesota 전산학 박사
1997～1999 Performance Analyst, Intel Corp.,
1999～현재 한양대학교 공과대학 전자전기컴퓨터공학부 부교수
관심분야 : 멀티미디어 시스템, 운영체제, 컴퓨터 네트웍
E-mail : yjwon@ece.hanyang.ac.kr

김 영 진

2005 평택대학교 이과대학 컴퓨터과학과 학사
2005～현재 한양대학교 공과대학 전자전기컴퓨터공학부 석사 재학중
관심분야 : 임베디드 시스템, 멀티미디어 시스템, 운영체제
E-mail : yjkim@ece.hanyang.ac.kr