대형 멀티미디어 파일을 위한 파일 시스템 구현

손 정수, 이 민석
한성대학교 컴퓨터공학부

An Implementation of a File System for Large Multimedia File

Jungsoo Sohn, Minsuk Lee

School of Computer Engineering, Hansung University

Abstract

멀티미디어 시스템에서는 통상적으로 매우 큰 크기의 파일이 저장되고 재생된다. 이 파일들은 읽기 중심이며 재사용 가능성이 낮아 기존의 파일 시스템들이 가정하는 형태의 파일이 아니기 때문에 이런 파일이 자주 사용되는 멀티미디어 시스템의 경우 기존 파일 시스템의 성능은 그리 좋지 않다.본 논문에서는 멀티미디어 시스템을 위한 새로운 파일 시스템을 구현하였다. 성능 측정 결과 새 파일 시스템이 기존 파일 시스템인 Ext2, Ext3 보다 쓰기, 읽기, 쓰기/읽기 동시 수행에 있어서 각각 39.75% ~ 40.67%, 36.48% ~ 43.36%, 28.04% ~ 32.60% 높은 성능을 나타내었다. 이 파일 시스템은 리눅스 상에서 구현되었으며 어렵지 않게 다른 운영 체제에도 적용이 가능하다.

In multimedia systems, very large files are stored and played. Because those files are read-centric and have low reusability, legacy file systems that are optimized for the use of many small files, do not show good performance in the multimedia systems. We implemented a new file system to deal with those characteristics. The new file system shows the superior performances in read, write and mixed operation to the Linux' s EXT2 or EXT3 file systems. The new file system was implemented on the Linux operating system, and it also can be easily ported to other operating systems.

Keywords : File System, Multimedia, Linux

논문접수일 : 2003년 8월 10일 논문게재확정일 : 2003년 10월 4일
※ 본 연구는 한성대학교 2002년도 교내 연구비 지원과제임

1. 서 론

최근, 멀티미디어 기술이 급격히 발전하면서 DVR, PVR, 홈서버, 멀티미디어 서버 등 대형 멀티미디어 파일을 저장하고 재생하기 위한 장치들이 많이 등장하고 있다. 또, 이러한 장치들이 주로 리눅스를 운영 체제로 사용하고 있다. 그러나 기존의 리눅스 파일 시스템들은 상대적으로 작은 파일의 잦은 입출력에 최적화되어 있어 대형 멀티미디어 파일을 다루는 시스템에는 성능과 안정성 면에서 적합하지 않다.

본 논문에서는 대형 멀티미디어 파일을 저장하고 재생하기 위한 새로운 멀티미디어 파일 시스템인 MLFS (Linux File System for Multimedia System)를 구현하였으며, 다양한 실험으로 성능 평가를 실시하여 MLFS의 성능을 검증하였다. 개발된 파일 시스템은 약간의 수정으로 리눅스가 아닌 다른 운영체제에도 적용 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 기존 리눅스 파일 시스템의 종류와 대형 멀티미디어 파일이 저장되는 시스템에서 기존 파일 시스템들의 문제점을 언급하며, 3장에서는 MLFS의 구조 및 구현 내용을 기술한다. 4장에서는 MLFS 대한 성능 평가를 실시하며 5장에서는 결과 분석을, 6장에선 본 논문의 결론을 기술함으로 본 논문을 마치고자 한다.

2. 관련 연구

2.1 리눅스 파일 시스템

파일 시스템은 컴퓨터의 하드 디스크에 데이터를 저장하고, 회수하는 운영체제를 위한 논리적인 수단이다. 즉, 파일의 자료구조, 데이터가 디스크에 저장되는 방식, 파일을 읽고, 쓰는 연산 등 파일에 관련된 모든 것을 관리한다 [Moshe Bar, 2001 ; 권수호, 2002].

리눅스 파일 시스템은 1991년 리눅스가 탄생한 이래 지난 수년 동안 지속적인 발전을 거듭해 왔다. 그 중 오늘날 가장 널리 사용되는 파일 시스템은 리눅스의 표준 파일 시스템인 EXT2와 저널링 파일 시스템들인 EXT3, JFS, XFS, ReiserFS이 있다.

2.1.1. EXT2

EXT2 (The Second Extended File System)는 Way Davidson에 의해 설계된 리눅스 표준 파일 시스템으로 일반 파일, 디렉토리, 디바이스 파일, 심볼릭 링크의 표준 Unix 파일 타입을 지원한다. 또한 4TB까지의 대용량의 파티션 지원이 가능하며 255 문자까지의 긴 파일 이름을 지정할 수 있고 필요에 따라 1012 문자까지 확장이 가능하다[Moshe Bar, 2001 ; Remy Card 외, 1994].

2.1.2 EXT3

EXT3 (The Third Extended File System)파일 시스템은 Stephen Tweedie가 설계한 것으로 EXT2에 저널링을 도입한 EXT2의 확장 파일 시스템이다[Stephen Tweedie, 2000]. EXT2와 똑같은 디스크 포맷과 메타 데이터를 가지고 있으며 메타 데이터 및 데이터 저널링을 지원하고 순차적인 파일 이름 찾기가 가능한 블록 기반이다[Daniel Robbins ; Ray Bryant외, 2002].

2.1.3 ReiserFs

ReiserFS는 Namesys의 Hans Reiser와 그가 이끄는 팀이 개발한 파일 시스템으로 작은 파일 엑세스 성능 향상에 초점을 맞추어 설계되었다. ReiserFS는 메타 데이터 저널링하며 디렉토리,

파일, 데이터를 구성하는 방법으로 B* 트리를 사용한다[Ray Bryant외, 2002].

2.1.4 JFS

JFS (Journaled File System)는 IBM 사에 의해 제작된 파일 시스템으로 현재 IBM의 엔터프라이즈 서버에서 사용되고 있다. JFS는 서버 환경에서의 높은 처리량을 위해 설계된 파일 시스템으로서 메타 데이터 저널링을 지원하며 메타 데이터를 B+ 트리로 관리한다. 또한, 동적 아이노드 할당 방법을 사용하여 사용되지 않는 아이노드는 해제하여 디스크 저장 공간을 절약할 수 있다[Ray Bryant외, 2002 ; 권우일외 ; 11].

2.1.5 XFS

XFS는 SGI 사에 의해 개발되었으며 IRIX 운영체제에 사용되고 있다[12]. XFS는 블록 크기를 512B에서 64KB까지 늘릴 수 있고, 이를 통해 대용량 파일을 저장하는 서버를 위해 설계되었으며 메타 데이터 저널링을 지원한다. 메타 데이터 관리를 위해 B* 트리를 사용하며 완전한 64-bit 파일 시스템으로 수백만 테라 바이트 용량을 지원한다[Moshe Bar, 2001 ; Ray Bryant ; Adam Sweeney].

2.2 대형 멀티미디어 파일을 위한 파일 시스템

2.2.1 멀티미디어 파일의 특징

멀티미디어 파일이라 함은 숫자나 문자 위주의 데이터를 벗어나 텍스트, 그래픽, 이미지, 오디오, 비디오, 애니메이션 등 여러 미디어 데이터 정보가 디지털 방식으로 융합된 형태의 파일을 말한다[4]. 멀티미디어 파일은 다음과 같은 특징을 갖는다.

1) 대형이다. 멀티미디어 파일은 주로 음악, 영상 등의 정보를 저장하므로 한 파일이 수십 메가 바이트에서 수기가 바이트까지 매우 큰 데이터로 유지된다.
2) 파일에 대한 읽기, 쓰기 연산이 주로 순차적으로 일어난다. 멀티미디어 파일은 여러 가지 데이터 형태가 복잡하게 얽혀 있다. 하지만 이 파일에 대한 연산은 빨리 감기, 재생, 빠른 재생, 저장 등 각각의 미디어 데이터에 대해 순차적 읽기, 쓰기 형태로 이루어진다.
3) 읽기 연산 중심이다. 대표적인 멀티미디어 파일인 영화, 애니메이션, 음악 파일들은 한번 디스크에 기록 된 후, 계속해서 읽혀 재생되는 읽기 연산이 대부분을 차지한다.
4) 멀티미디어 데이터는 특별한 형식이 없는 비정형의 구조를 갖는다. 이미지 데이터의 경우 일정한 구조를 갖지 않는 일정 비트의 연속적인 스트림(stream)으로 볼 수 있다.
5) 시간성을 갖는다. 오디오와 비디오, 애니메이션 데이터는 본질적으로 시간에 의존하는 특성을 가지며, 여러 가지 멀티미디어 데이터가 복합적으로 시간적인 순서에 따라 표현되는 동기화 특성이 존재한다.

이중 파일 시스템의 성능과 관련이 있는 것은 1),2),3) 이다. 나머지 것들은 응용프로그램에서 멀티미디어 파일에 대한 동기화와 관리 기법과 연계가 있다. 따라서 본 연구에서는 멀티미디어 파일 가운데 파일 시스템의 성능과 관련이 있는 1),2),3)의 특성에 대해 고려하며 분석하고 이를 지원하기 위한 기존 파일 시스템의 문제점을 제시하고자 한다.

2.2.2 대형 멀티미디어 파일에 대한 기존 파일 시스템의 문제점 (EXT2, EXT3)

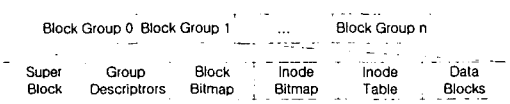
현재 리눅스에서 가장 많이 사용되고 있는 파일 시스템으로는 EXT2와 EXT3이 있으며 이들이 대형 멀티미디어 파일을 지원하는데 가지는 문제점은 다음 세 가지로 요약할 수 있다.

1) 디스크 상의 데이터 배치로 인한 문제
2) 데이터의 트리 구조 관리로 인한 문제
3) 버퍼 캐쉬 사용으로 인한 문제

멀티미디어 데이터를 효과적으로 읽고 쓰기 위해서는 디스크 헤드의 이동 시간(seek time)을 최소화하는 것이 가장 중요하다. 디스크 헤드의 이동 시간은 데이터 블록의 배치 방식과 메타 데이터의 구조에 따라 결정되며 이는 파일 시스템에 의해 정의된다[원유집외]. 따라서 멀티미디어 시스템을 위한 파일 시스템은 멀티미디어 파일의 특성을 감안한 데이터 블록 배치 방식과 이를 반영한 메타 데이터의 구조를 갖도록 설계되어야 한다. 그러나 유닉스 파일 시스템에 많은 영향을 받은 EXT2는 텍스트 기반의 비교적 작은 파일의 처리 성능 향상을 중심으로 발전하였다. 이로 인해 개인용 멀티미디어 시스템에 적용하는데 심각한 성능 상의 문제점이 발생하게 된다.

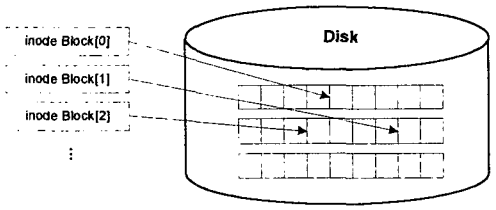
2.2.2.1 디스크 상의 데이터 배치 방법에 의한 문제

EXT2는 (그림 1)과 같은 디스크에 데이터 블록을 배치한다.



(그림 1) EXT2의 디스크 데이터 배치

EXT2는 디스크 헤드의 이동 시간을 줄이기 위해 블록 그룹 메커니즘을 사용하여 데이터 블록들을 관리한다. 블록 그룹은 (그림 2)와 같이 일련의 연속된 실린더들의 그룹으로 데이터 블록을 상대적으로 좀더 가까운 실린더 위치에 배치하여 디스크 탐색의 효율성을 향상시키는 메커니즘이다. 이는 사이즈가 작은 여러 파일들을 관리할 경우 시스템의 성능을 효과적으로 증가시킨다. 그러나 파일의 대부분이 블록 그룹의 사이즈보다 훨씬 크고 높은 순차적 접근 특성을 가진 멀티미디어 파일의 경우에도, 파일은 여러 개의 다른 블록 그룹으로 나누어져 저장되며 데이터 블록은 디스크 상에 분산되게 된다. 이로 인해 디스크 헤드는 좀더 빈번히 움직이게 되어 디스크 탐색 오버헤드가 발생한다[원유집외].

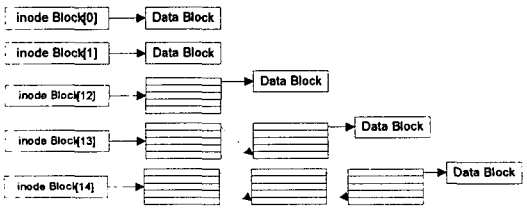


(그림 4) EXT2의 블록 기반 할당 구조

2.2.2.2 데이터의 트리 구조 관리로 인한 문제

EXT2의 블록 그룹은 슈퍼 블록, 그룹 디스크립터, 비트맵, 아이노드 테이블, 데이터 블록으로 구성되어 있다. 슈퍼 블록은 EXT2 파일 시스템을 유지하고 운용하는데 이용되는 매우 중요한 정보이며, 그룹 디스크립터는 전체 블록 그룹들의 상태를, 아이노드 비트맵은 해당 블록 그룹 내에 존재하는 아이노드의 사용 유무를, 아이노드 테이블은 블록 그룹 내 여러 아이노드를 지정하며 데이터 블록은 실제 데이터에 대한 논리적 모델을 나타낸다. 이중 데이터의 관리 구조를 결정하는 것이 아이노드이다. 아이노드

는 EXT2 파일 시스템에서 데이터 블록들에 대한 포인터 및 관련 정보를 가지는 것으로 EXT2는 이 아이노드를 통해 (그림 3)과 같은 트리 구조로 파일 시스템 내 모든 객체를 관리한다. 이러한 트리 구조는 작은 데이터 블록이 여러 군데 흩어져 있을 경우 데이터 탐색에 있어서 매우 효과적일 수 있다. 그러나 순차적 읽기 쓰기를 수행하는 멀티미디어 파일의 경우 불필요한 디스크 검색 오버헤드를 발생시켜 시스템의 성능 저하를 가져오는 원인이 된다. 또한 데이터 블록을 관리하기 위해 아이노드와 같은 메타 데이터를 많이 유지할 경우 그 만큼 많은 디스크 I/O를 필요로 하게 되어 시스템의 성능을 떨어드린다[Moshe Bar, 2001].



(그림 3) EXT2의 트리 구조

2.2.2.3 버퍼 캐쉬 사용으로 인한 문제

멀티미디어 데이터를 위한 EXT2의 또 다른 문제는 버퍼 캐쉬의 사용으로 인해 발생한다. 버퍼 캐쉬는 데이터의 재사용 가능성을 고려하여 메모리의 일정 부분을 캐쉬로 사용하는 정책이다. 이는 서버와 같이 많은 프로그램이 같은 파일을 재사용할 때 시스템의 성능을 크게 향상시켰다. 그러나 단일 프로그램이 순차적으로 데이터에 엑세스하는 멀티미디어 시스템의 경우 오히려 메모리 낭비와 많은 스왑핑으로 인한 시스템의 성능 저하를 가져오는 원인이 된다. 또한, 서버와 달리 불시에 전원을 끄는 사례가 빈번하게 발생하는 가전 형태의 멀티미디어 시스템의 경우 미처 디스크에 기록되지 못한 많은 데이터가 한 순간에 깨져 데이터의 무결성을 보장하지 못하는 원인이 된다. 심각한 경우 파일 시스템 전체가 깨질 수 있고 복구에도 오랜 시간이 걸리게 된다.

로그 파일 시스템[Christian] 기반인 ReiserFS, XFS, JFS와 같은 저널링 파일 시스템을 사용하면 복구 속도와 안정성, 그리고 XFS의 경우 순차적 연산 능력은 어느 정도 올라가지만 저널링을 위한 메타 데이터의 주기적 기록 때문에 시스템 성능이 떨어진다. 실제로 권우일 등이 행한 연구에서 DVR과 같은 멀티미디어 환경을 대상으로 시험한 결과, 저널링 파일 시스템들은 EXT2보다 전반적으로 낮은 읽기 성능을 보였다[권우일외].

위와 같은 문제는 기존 리눅스 파일 시스템이 대용량, 순차적 쓰기/읽기, 읽기 연산 중심이라는 멀티미디어 파일의 특성과 제한된 응용 프로그램 및 불시 전원 차단이라는 개인용 시스템의 특징을 충분히 고려하지 못했기 때문이며 이는 기존 리눅스 파일 시스템이 대형 멀티미디어 파일을 주로 사용하는 시스템에 적합하지 않음을 보여준다.

이 논문에서는 대형 멀티미디어 파일을 저장하고 플레이하기 위한 새로운 파일 시스템을 설계, 구현하였다.

3. 대형 멀티미디어 파일을 위한 리눅스 파일 시스템, MLFS의 구현

3.1 멀티미디어 파일 시스템의 요구 사항

멀티미디어 시스템을 위한 파일 시스템이 갖

추어야 할 두 가지 중요한 요구 사항은 아래와 같다.

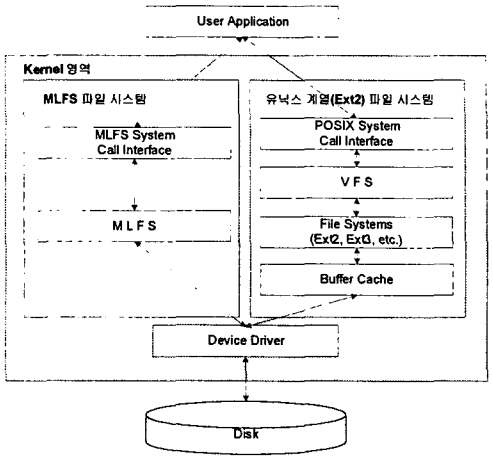
1) 대형 멀티미디어 파일을 위한 대용량의 블록 사이즈 구조
2) 순차적 데이터의 빠른 탐색을 위한 단순한 인덱스 구조
3) 빠른 읽기/쓰기 성능
4) 불시의 전원 차단에도 안전한 데이터 무결성

본 연구에서는 대형 멀티미디어 파일을 처리하기 위하여 기존 파일 시스템들의 단점을 보완하고 위 요구 사항을 만족하는 새로운 파일 시스템인 MLFS을 제안한다.

3.2 MLFS의 구조

3.2.1 MLFS의 전체 구조

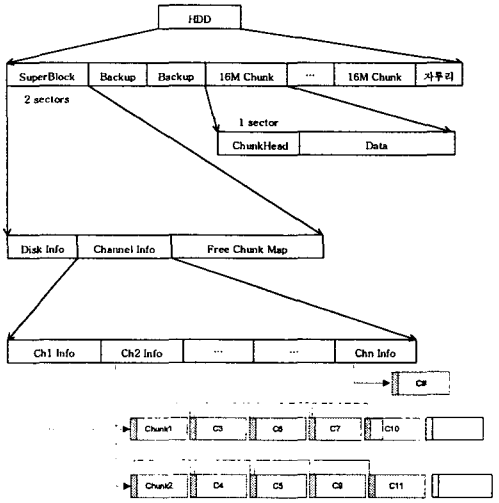
운영 체제에서 MLFS의 전체 구조는 (그림 4)와 같다. MLFS는 유닉스 계열 파일 시스템과 달리 자체 시스템 콜 인터페이스를 제공한다. 응용 프로그램은 POSIX와 거의 유사한 API 라이브러리를 통해 MLFS에 접근하여 파일에 대한 연산을 수행할 수 있다. 이 방법은 응용 프로그램의 호환성 면에서는 단점일 수 있으나, 범용 시스템이 아닌 멀티미디어 시스템에서는 크게 문제되지 않는다. 또 일반 유닉스 계열 파일 시스템은 디스크 블록에 대한 재사용을 대비한 성능 향상을 위해 버퍼 캐쉬를 파일 시스템과 디스크 사이에 두게 되며 데이터를 디스크에 직접 쓰지 않고 캐쉬를 통하여 관리한다. 이는 불시 전원차단 시 데이터의 무결성을 보장하지 못하는 원인이 된다. 그러나 MLFS는 최소한의 독립된 버퍼를 유지하며 대부분의 읽기 쓰기를 디스크에 바로 수행한다.



(그림 4) MLFS의 전체 구조

3.2.2 디스크 상의 데이터 배치

MLFS는 순차적 쓰기/읽기 특징의 멀티미디어 특성을 반영하여 (그림 5)와 같은 데이터 배치 구조를 갖는다.



(그림 5) MLFS의 디스크 데이터 배치

MLFS는 하드 디스크를 하나의 슈퍼 블록과 이를 위한 백업 그리고 최소 4MB, 최대 16MB Chunk들로 구성한다. 슈퍼 블록은 파일 시스템

전체 상황과 정보를 유지하는 중요한 데이터로 디스크의 시작 섹터에 저장되며 만일의 사태에 대비하여 두 번 백업을 한다. 슈퍼 블록에 기록되는 데이터는 <표 1>과 같다.

〈표 1〉 슈퍼 블록 데이터 구조

FS_ID	파일 시스템 아이디
ChunkSize	Chunk의 사이즈
Chunk1Start	첫번째 Chunk위치
NumChunk	전체 Chunk 수
NumFreeChunk	빈 Chunk 수
ChannelInfo[채널수]	채널 정보
Checksum	슈퍼 블록 체크 섬
FCB[FCB_SIZE]	Free Chunk 비트맵

슈퍼 블록에는 <표 2>와 같은 채널 정보를 채널 수만큼 유지한다.

〈표 2〉 채널 데이터 구조

MediaType	저장된 데이터 종류
ChunkCount	채널에 할당된 Chunk 수
StartTime	채널의 첫 데이터의 Time Stamp
FirstSeq	첫 번째 Chunk의 순서 번호
LastSeq	마지막 Chunk의 순서 번호
FirstChunk	첫 번째 Chunk의 번호
SecondChunk	두 번째 Chunk의 번호
LastChunk	마지막 Chunk의 번호
Last-1Chunk	끝에서 두 번째 Chunk의 번호
DataSize	채널의 전체 데이터 크기

채널은 MLFS가 데이터를 관리하는 단위로 대부분의 파일 시스템들은 파일을 유지하며 파일에 대해 읽기 쓰기를 수행하는 것과 같이 MLFS에는 데이터를 채널 단위로 관리하며 사용자는 파일을 사용할 때와 같이 해당 채널을 열고 채널에 쓰기/읽기를 수행하게 된다. 이 채널에 대한 데이터 구조는 슈퍼 블록 내에 존재하며 채널 수는 파일 시스템을 생성할 때 사용자가 지정 결정한다.

또, MLFS는 4MB, 8MB 또는 16MB 크기로 관리되는 Chunk 블록을 사용한다. Chunk 블록은 Chunk에 대한 정보를 가진 Chunk 헤더와 실제적인 데이터로 구성된다. Chunk 헤더는 Chunk의 첫번째 섹터에 저장되며 유지하는 데이터는 <표 3>과 같다.

〈표 3〉 Chunk 헤더 구조

START_CODE	Chunk의 시작을 알리는 코드
ChunkNumber	Chunk의 물리적인 Chunk번호
Channel	이 Chunk를 사용하는 채널 번호
PrevChunk	같은 채널의 앞 Chunk 번호
NextChunk	같은 채널의 다음 Chunk 번호
PPrevChunk	같은 채널의 앞 앞 Chunk 번호
NNextChunk	같은 채널의 다음 다음 Chunk 번호
SeqNumber	이 채널에서의 일련 번호
Offset	실제 데이터가 시작되는 위치
Length	기록된 데이터의 크기
CheckSum	Chunk 헤더 체크 섬

실제 각 Chunk의 데이터는 두 번째 디스크 섹터에서 시작하며 하나의 Chunk에 기록할 수 있는 최대 데이터 양은 Chunk가 16MB인 경우, 16MB - 512B = 16,776,704 바이트이다.

3.2.3 대용량의 연속 블록 크기 구조

MLFS는 4MB, 8MB, 16MB 크기의 연속 블록 크기를 지원한다. EXT2는 파일 시스템 생성 시 블록의 크기를 1KB, 2KB, 4KB 가운데 하나로 설정할 수 있다. 그러나 이같이 상대적으로 작은 블록 크기는 대형 멀티미디어 파일을 저장할 때 파일이 많은 블록에 디스크 상에 흩어져 디스크 탐색 시간을 증가시키는 원인이 되었다. MLFS는 대용량의 멀티미디어 파일을 효과적으로 저장하기 위해 Chunk를 4MB, 8MB, 16MB 크기로 설정할 수 있도록 설계

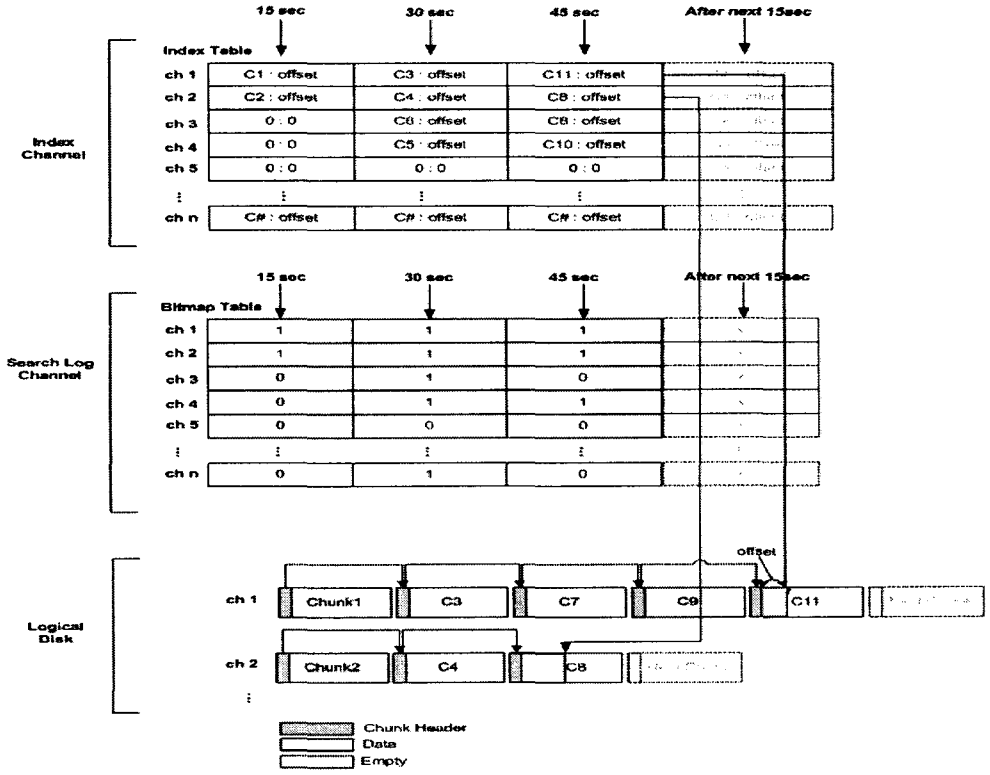
되었으며 이를 통해 보다 많은 데이터를 연속적으로 유지하여 데이터의 분산을 막았다. 또한 이 같은 대용량의 단위 데이터 크기 구조는 기존 파일 시스템보다 메타 데이터 수와 디스크 I/O 수를 크게 줄여 시스템의 성능 향상을 가져온다.

하지만 한 블록 크기를 4MB, 8MB, 16MB의 대형 블록 단위(Chunk)로 유지하면 내부 단편화 문제가 커지게 된다. 만일 10KB의 작은 크기의 데이터를 저장할 경우 나머지 15MB의 공간을 사용할 수 없게 되는 매우 심각한 문제가 발생한다. 그러나 대형 멀티미디어 시스템의 경우 불과 몇 초 만에 수백 메가 바이트를 디스크에 순차적으로 기록하기 때문에 MLFS의 대부분의 Chunk는 꽉 차게 되며 단편화 문제는 거의 발생하지 않는다.

MLFS의 Chunk 크기는 파일 시스템을 생성시 설정할 수 있으며 이후에는 임의로 바꾸는 것이 불가능하다. MLFS를 위한 성능측정에서 실제 Chunk 크기를 변경하여 실험을 실시하였다.

3.2.4 MLFS의 인덱스 구조

MLFS는 유닉스 계열 파일 시스템과 달리 아이노드를 통한 트리 구조를 사용하지 않고 (그림 6)과 같이 매우 단순한 인덱스 구조를 가진다. 인덱스 구조를 단순화한 이유는 아이노드를 이용한 트리 구조를 사용할 경우 많은 양의 메타 데이터를 유지해야 할 뿐 아니라 데이터를 순차적으로 읽는 경우에도 오버헤드를 발생시키기 때문이다. 실제 인덱스 구조는 응용 프로그램에 따라 다르게 만들 수 있으나 그



(그림 6) MLFS의 디스크 데이터 배치

림에서는 매 15초마다 각 채널의 데이터 위치를 기록하는 DVR의 경우를 묘사한 것이다.

MLFS는 순차 인덱싱과 임의 위치 인덱싱 모두 가능하다. 순차 인덱싱은 (그림 5)에서 보듯이 슈퍼 블록 a 해당 채널 a 첫번째 Chunk로 접근하여 이후 Chunk를 순차적으로 엑세스하는 것으로 이루어진다. 반면 임의 위치 인덱싱을 위해서는 별도의 두개의 채널을 가지고 있다. 그 중 인덱스 채널은 각 채널 별로 데이터의 위치를 Chunk 번호와 옵셋을 사용하여 유지한다. 여기서 Chunk 번호는 디스크 상에서의 물리적 Chunk 번호이며 옵셋은 Chunk 내에서 데이터의 위치를 지정한다. 또 다른 한 채널은 인덱스 구간 ((그림 6)에서 15초) 동안 각 채널에 데이터가 기록되었는지를 비트맵으로 나타낸다.

만일 데이터가 쓰였다면 비트맵 필드는 1이 되며, 데이터가 쓰여지지 않았다면 0이 된다. 응용 프로그램은 인덱스 채널과 Search Log 채널을 열고 주기적으로 디스크에 인덱스 정보를 기록한다. 따라서 응용 프로그램은 채널 정보와 상대적 시간 정보를 가지고 원하는 데이터를 엑세스할 수 있다. 인덱스 채널을 사용하지 않으면 임의 위치 엑세스가 불가능하게 되지만 각 채널에 대한 순차 엑세스는 아직도 가능하다. 읽기는 임의, 순차 방식을 모두 지원하지만 쓰기의 경우 MLFS에서는 순차 쓰기만 지원한다.

인덱스 채널과 Search Log 채널의 기록 간격은 파일 시스템을 생성 시 사용자가 결정한다

3.2.5 순차 블록을 통한 순차 접근 구조

MLFS는 블록 그룹을 사용하지 않는다. 대신 순차적 연산을 위해 16MB(또는 4MB, 8MB)크기의 Chunk에 데이터를 순차적으로 기록한다. 따라서 디스크 헤드는 대용량의 Chunk를 순차 방향으로 접근하게 되며 디스크 탐색 시간은 현저하게 줄어들게 된다.

3.2.6 데이터 무결성 보장

MLFS는 데이터를 버퍼 캐쉬를 통하지 않고 디스크에 바로 기록하는 방법을 통해 불시의 전원 차단에도 높은 데이터 안정성을 보인다. 가전용으로 만들어지는 많은 멀티미디어 시스템은 사용자가 불시에 전원을 끄는 경우를 빈번하게 당하게 될 가능성이 높으며, 버퍼 캐쉬를 사용하는 경우 전원 차단과 함께 미처 디스크에 기록되지 못한 데이터를 잃게 되어 파일 시스템이 깨지게 된다. MLFS는 데이터를 디스크에 기록할 때 버퍼 캐시를 통하지 않고 바로 디스크에 쓰도록 하여 최근의 데이터를 디스크에 저장하며, 간단한 인덱스 구조 때문에 불시의 전원 차단 후에 필요한 파일 시스템 검사 시간이 매우 짧다.

3.2.7 미리 읽기

MLFS는 읽기 속도를 높이고자 효과적인 미리 읽기(Read-ahead)를 수행한다. 멀티미디어 시스템은 읽기 위주의 연산을 수행한다. 이는 디스크에 쓰는 속도보다 디스크로부터 읽는 속도가 더 중요함을 말해준다. 따라서 MLFS는 이를 위해 미리 읽기를 기존 파일 시스템보다 효과적으로 수행함으로써 읽기 속도가 빨라졌다. MLFS의 경우 디스크로부터 읽기를 수행하면 32KB 데이터를 읽을 때마다 최대 16*32KB 양의 데이터를 읽기 버퍼에 유지한다. 따라서 일단 읽기를 수행하면 다음의 읽기를 위해 이미 충분한 데이터를 버퍼에 유지하게 되므로 읽는 시간이 획기적으로 줄어들게 된다.

4. MLFS의 성능 평가 실험

본 실험에서는 EXT2, EXT3을 MLFS와의 성능 평가 비교 대상으로 선정하였으며 성능 평가 기준은 시험 프로그램에 대한 응답 시간으로 하였다.

4.1 실험 환경

성능 평가를 실시할 시스템의 환경은 <표 4>와 같다.

〈표 4〉 시스템 환경

CPU	MIPS 4Kc (300Mhz)
Memory	32MB
IDE Interface	DMA 66
Linux Kernel	2.4.18

4.2 파일 시스템 부하 설정

파일 시스템의 성능을 측정하기 위해 사용된 부하는 <표 5>와 같다.

〈표 5〉 성능 시험 용 부하

버퍼 크기	32KB
반복 회수	320(회)
단일 쓰레드 W/R 크기 (버퍼크기) * (Loop count)	10MB
쓰레드 수	1개, 4개, 8개

실제 쓰여지는 데이터는 메모리 상에 존재하는 32KB의 데이터이다. 테스트 프로그램은 쓰레드 수를 1개, 4개, 8개로 증가시키며 테스트를 수행하며 버퍼 크기(32KB)만큼 해당 파일(MLFS의 경우는 해당 채널)에 쓰기, 읽기를 수행하게 된다. 총 쓰기/읽기 되는 데이터 크기는 쓰레드 수에 비례하여 10MB, 40MB, 80MB가 된다.

4.3 테스트 프로그램 구현

테스트 프로그램은 멀티미디어 파일에 대한 성향을 잘 반영하고 있어야 한다. 개인용 멀티미디어 시스템의 경우 주로 순차적인 쓰기, 읽기 연산이 대부분이기 때문에 테스트 프로그램도 이와 같은 기능을 갖도록 구현되었다. <표 6>은 테스트 프로그램의 동작 방법에 대해 기록하고 있다.

〈표 6〉 테스트 프로그램 동작 방법

쓰기	*. EXT2, EXT3 1. 버퍼 크기만큼 메모리를 할당받아 채운다. 2. 각 쓰레드는 해당 파티션에 자신의 파일을 연다. 3. 각 쓰레드는 반복 회수만큼 돌며 버퍼에 있는 데이터를 해당 파일에 버퍼 크기씩 쓴다.
	*. MLFS 1. 버퍼 크기만큼 메모리를 할당받아 채운다. 2. 각 쓰레드는 해당 채널을 연다. 3. 각 쓰레드는 반복 회수만큼 돌며 버퍼에 있는 데이터를 해당 채널에 버퍼 크기씩 쓴다.
읽기	*. EXT2, EXT3 1. 버퍼 크기만큼 메모리를 할당 받는다. 2. 각 쓰레드는 해당 파티션에 자신의 파일을 연다. 3. 각 쓰레드는 반복 회수만큼 돌며 해당 파일로부터 버퍼 크기씩 읽어 버퍼에 저장한다.
	*. MLFS 1. 버퍼 크기만큼 메모리를 할당 받는다. 2. 각 쓰레드는 해당 채널을 연다. 3. 각 쓰레드는 반복 회수만큼 돌며 해당 채널로부터 버퍼 크기씩 읽어 버퍼에 저장한다.

4.4 측정 방법

성능 측정을 수행할 때 시간 측정 방법과 버퍼 캐쉬의 효과를 없애는 전략이 고려되어야 한다. 특히, 버퍼 캐쉬를 통해 디스크 I/O가 일어나면 파일 시스템에 대한 정확한 성능 평가가 이루어질 수 없기 때문에 이에 대한 선행 처리를 해주는 것이 필요하다.

본 실험에서는 시간 측정을 위해 사용한 gettimeofday()는 리눅스 커널에서 제공하고 있는 함수로 마이크로 초 단위까지 측정이 가능하다. 측정 방법은 이 함수를 디스크 I/O를 수행하는 쓰레드들의 생성과 소멸 앞 뒤에 추가하여 쓰레드의 시작과 끝의 시간 차를 구하였다

버퍼 캐쉬의 효과를 없애는 일반적인 방법으로는 시스템에 메인 메모리보다 큰 파일을 쓰는 방법과 O_DIRECT를 사용하는 방법, 시스템을 재부팅하는 방법 등이 있다[Ray Bryant외]. 본 실험에서는 실험 전 시스템을 재부팅하는 것을 통해 버퍼 캐쉬의 효과를 없앴다. 실험 순서는 <표 7>과 같다

〈표 7〉 성능 평가 실험 순서

1차				2차			
MLFS Chunk 크기별 평가				파일 시스템별 평가			
4MB	W	R	W/R	EXT2	W	R	W/R
8MB	W	R	W/R	EXT3	W	R	W/R
16MB	W	R	W/R	MLFS	W	R	W/R

성능 평가를 위한 실험은 두 단계로 이루어졌다. 1 단계 실험은 MLFS의 Chunk 크기에 따른 성능 변화를 평가하여 가장 높은 성능을 보이는 Chunk 크기를 결정하는 실험으로 Chunk 크기를 4MB, 8MB, 16MB로 변경하며 각각 쓰기, 읽기, 쓰기/읽기 동시 수행에 대한 성능 평가를 실시하였다. 2 단계 실험은 파일 시스템 별 평가로 1 단계 실험에서 결정된 Chunk 크기를 적용한 MLFS와 EXT2, EXT3과의 성능 비교 평가를 실시하였다.

이때, 각각의 실험들에 대해 쓰레드 수를 1개, 4개 8개로 변경하며 성능을 측정하였으며, 쓰레드 수가 1개인 경우 쓰기/읽기 동시수행에 대한 평가를 수행하지 않았다. 4개, 8개의 쓰레드 수행에 있어서는 쓰기/읽기의 쓰레드 비율을 1:4로 잡았는데 이는 멀티미디어 시스템이 읽기 위주라는 특징을 반영한 것이다.

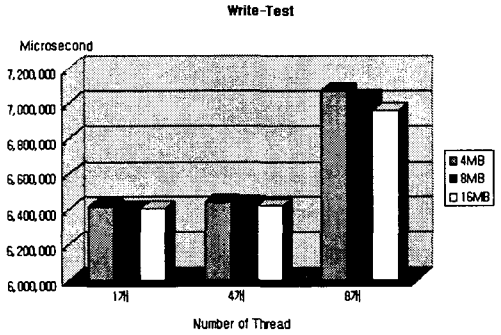
5. 결과 분석 및 평가

본 장에서는 앞서 정의된 시험용 부하를 이용해 성능 평가를 실시하였다. 평가는 1 단계, 2 단계로 나누어 실시하였으며 1 단계 실험은 MLFS의 Chunk 크기 별 평가를, 2 단계 실험은 1 단계 실험 결과로 알게 된 최적 Chunk 크기를 반영한 MLFS와 다른 리눅스 파일 시스템과의 평가를 실시하였다. 2 단계 성능 평가에서는 EXT2와 EXT3을 MLFS와의 성능 비교 대상으로 선정하였으며 성능 평가 기준은 시험 프로그램에 대한 응답 시간으로 하였다.

5.1 MLFS Chunk 크기 별 성능 평가

MLFS에 적용할 최적의 Chunk 크기를 결정하기 위해 4MB, 8MB, 16MB로 Chunk 크기를 변경하며 쓰기, 읽기, 쓰기/읽기 동시수행에 대한 성능을 측정하였다.

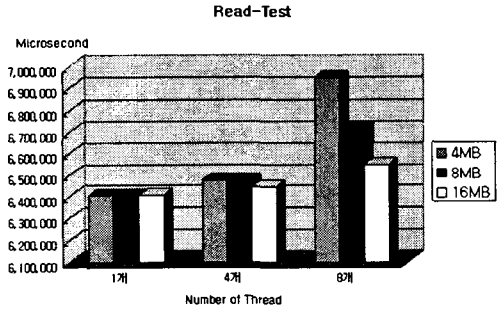
쓰기에 대한 성능 측정 결과는 (그림 7)과 같다.



(그림 7) MLFS의 Chunk 크기 별 쓰기 성능 측정 결과

쓰기의 경우 쓰레드 수가 1개, 4개일 경우 0.3%미만으로 세 경우 거의 차이가 없어 보인다. 그러나 쓰레드 8개를 동시에 돌렸을 경우 16MB가 4MB, 8MB보다 각각 1.6%, 1% 높게 나타났다. 이는 Chunk 크기가 크면 클수록 한 Chunk 당 기록할 수 있는 양이 증가하여 보다 적은 Chunk를 사용하므로 free Chunk를 가져오는데 걸리는 시간을 줄일 수 있기 때문이다.

읽기에 대한 성능 평가 결과는 (그림 8)와 같다.



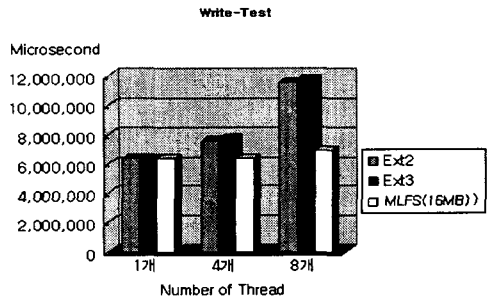
(그림 8) MLFS의 Chunk 크기 별 읽기 성능 측정 결과

읽기의 경우 쓰레드 수가 1개, 4개일 경우 쓰기와 마찬가지로 거의 차이가 없어 보인다. 그러나 쓰레드 수가 8개의 경우 16MB가 4MB, 8MB보다 각각 5.71%, 2.48% 높게 나타났다. 이는 미리 읽기 정책이 같다해도 사이즈가 큰 Chunk 일수록 보다 많은 데이터를 읽게 되므로 상대적으로 적은 수의 Chunk에 접근하게 된다. 따라서 Chunk를 찾는데 드는 시간이 줄어들게 되기 때문이다. 이 같은 결과는 가장 높은 성은을 보인 MLFS의 Chunk 크기는 16MB임을 나타낸다.

5.2 파일 시스템별 성능 평가

여기서는 5.1의 실험 결과 가장 성능이 우수한 16MB(Chunk 크기) MLFS와 EXT2, EXT3의 성능 평가를 실시하였다.

쓰기에 대한 성능 측정 결과는 (그림 9)과 같다.

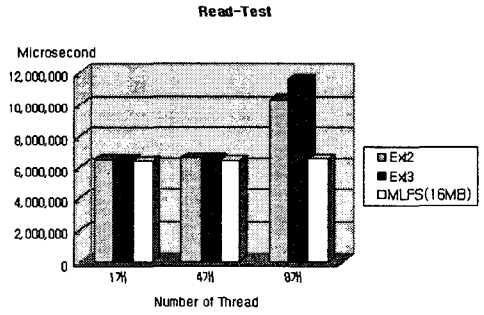


(그림 9) 파일 시스템 별 쓰기 성능 측정 결과

쓰기의 경우 쓰레드 수가 1개일 경우 차이가 없으며 4개의 경우 MLFS가 EXT2, EXT3 보다 각각 15.42%, 17.16% 성능이 높은 것으로 나타났다. 또한 8개의 경우 각각 39.75%, 40.67% 성능이 높은 것으로 나타났다. 쓰기 성능이 높은 이유는 MLFS가 EXT2, EXT3 보다 훨씬 적은 양의 메타 데이터를 쓰므로 디스크 헤드의 움직임으로 인한 성능 저하가 적기 때문이다.

또, 쓰레드 수가 증가할수록 성능 차이가 크게 나타나는 것은 EXT2, EXT3의 경우 쓰레드 1개일 때 데이터를 비교적 근접한 블록에 기록하나 프로세스가 많아 지고, 데이터가 많아지면서 데이터가 디스크 상에 분산 되어 디스크 헤드의 이동 시간이 증가하기 때문이다. 반면 MLFS의 경우 16MB Chunk에 데이터를 순차적으로 기록하므로 데이터 분산이 줄어들기 때문에 높은 성능을 보인다.

읽기에 대한 성능평가 결과는 (그림 10)와 같다.



(그림 10) 파일 시스템 별 읽기 성능 측정 결과

읽기의 경우 쓰레드 수가 1개일 경우 MLFS가 EXT2, EXT3보다 1.94%, 1.89%, 4개일 경우 2.47%, 2.08%, 8개일 경우 36.48%, 43.36% 높은 성능을 보였다. MLFS의 읽기 성능이 높은 이유는 EXT2와 EXT3은 작은 블록들을 트리 구조를 통해 접근하지만 MLFS는 훨씬 큰 순차 블록을 엑세스하여 디스크 검색 속도를 현저히 감소시키기 때문이다. 또한, MLFS가 미리 읽기를 보다 많이 효과적으로 하여 디스크 I/O 수를 감소시키기 때문이다.

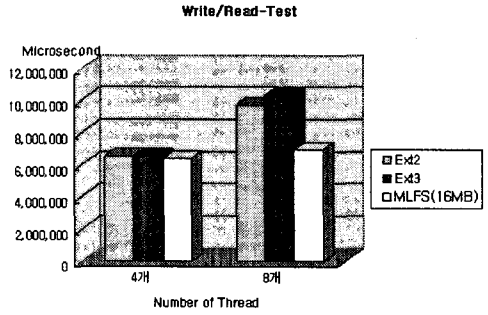
쓰기/읽기 동시 수행에 대한 성능 측정 결과는 <표 8>과 같다.

〈표 8〉 파일 시스템 별 쓰기/읽기 성능 측정 결과

(단위: Microsecond)

FS \ Thread	4개	8개
EXT2	6,576,667	9,690,000
EXT3	6,550,000	10,346,667
MLFS(16MB)	6,456,667	6,973,333

쓰기/읽기 동시 수행의 경우 쓰레드가 4개일 경우 MLFS가 EXT2, EXT3보다 각각 1.82%,1.42%, 8개일 경우 28.04%, 32.60%를 나타냈다. (그림 11)는 위의 결과를 도식화한 것이다.



(그림 11) 파일 시스템 별 쓰기/읽기 성능 측정 결과

6. 결론

기존 리눅스 파일 시스템은 대용량 파일 처리 미흡, 느린 읽기 속도, 부적절한 인덱스 구조로 인하여 대형 멀티미디어 파일이 저장되고 재생되는 멀티미디어 시스템을 지원하기 어려웠다. 본 연구에서는 위와 같은 문제점을 해결할 수 있는 새로운 파일 시스템인 MLFS를 설계하고 구현하였다.

MLFS는 데이터의 순차 관리, 적은 메타 데이터, 채널별 데이터 관리, 대용량 Chunk의 사

용, 효과적인 미리 읽기 등을 통해 대형 멀티미디어 데이터 저장을 위한 파일 시스템의 성능과 안정성을 확보하였다. 또한 성능 평가를 통해 MLFS의 최적 Chunk 크기가 16MB이며 MLFS가 EXT2, EXT3보다 안정성, 성능 면에서 뛰어남을 검증하였고 이를 통해 MLFS가 멀티미디어 파일 시스템으로서 적합함을 보였다. 이 같은 결과는 멀티미디어 시스템의 경우 MLFS를 사용하는 것이 시스템의 성능과 안정성을 크게 높일 수 있는 대안임을 말해준다.

MLFS의 구현은 앞으로도 지속적인 발전이 예상되는 멀티미디어 시스템을 위한 파일 시스템의 구현을 위한 좋은 기초 자료가 되며, 리눅스나 그 밖의 운영 체제를 사용하는 멀티미디어 상품에 쉽게 적용될 수 있다.

앞으로 서비스 품질(QoS)를 위한 고려가 파일 시스템 구현에 추가되어야 할 것이다.

참 고 문 헌

[1] 권수호, Linux Kernel Programming Bile 2nd Edition, 글로벌, 2002

[2] 권우일, 윤미현, 이동준, 장재혁, 양승민, DVR 시스템을 위한 저널링 파일 시스템의 성능평가, 한국정보과학회지

[3] 원유집,박진연, "정보가전용 멀티미디어 파일 시스템 기술"

[4] 멀티미디어와 정보화사회, 한국과학기술진흥재단, http://211.40.179.13/book_file/ke28/ke028-index.htm

[5] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck, "Scalability in the XFS File System", 1996 USENIX conference, 1996 http://oss.sgi.com/projects/xfs/papers/xfs_usenix/#fn1

[6] Christian Czezatke and M. Anton Ertl, "A Log-Structured Filesystem for Linux" 2000 USENUX Annual Technical Conference, 2000

[7] Dr. Stephen Tweedie, "EXT3, Journaling Filesystem", the Ottawa Linux Symposium, html document of OLS, 20 July, 2000 http://olstrans.sourceforge.net/release/OLS2000-EXT3/OLS2000-EXT3.html

[8] Moshe Bar, Linux File Systems, McGraw-Hill Companies, 2001

[9] Remy Card, Theodore Ts'o, and Stephen Tweedie ,"Design and Implementation of the Second Extended Filesystem", First Dutch International Symposium on Linux, ISBN 90-367-0385-9, 1994 http://e2fsprogs.sourceforge.net/EXT2intro.html

[10] Ray Bryant, Ruth Forester and John Hawkes, "Filesystem Performance and Scalability in Linux 2.4.17," 2002 USENIX Annual Technical Conference, 2002

[11] Journaled File System Technology for Linux, html document of IBM, http://oss.software.ibm.com/jfs/

[12] XFS html document of SGI, http://oss.sgi.com/projects/xfs/

[13] Daniel Robbins, "고급 파일 시스템 개발자 가이드 , Part 7 EXT3", IBM, http://www-903.ibm.com/developerworks/kr/linux/library/l-fs7.html

□ 저자소개

* 손 정 수

저자는 2002년 한성대학교 산업공학과 학사, 2003년 현재 한성대학교 컴퓨터공학과 석사 과정 재학 중

관심 분야 : 실시간 임베디드 시스템, 리눅스 파일 시스템

* 이 민 석

저자는 1986년 서울대학교 컴퓨터공학과 학사, 1988년 서울대학교 컴퓨터공학과 석사. 1995년 서울대학교 컴퓨터공학과 박사. 1999년-2002년 (주)팜팜테크 CTO, 1995년-현재 한성대학교 컴퓨터공학부 부교수 재직중, 관심 분야 : 실시간 시스템, 임베디드 시스템, 임베디드 리눅스

◈ 이 논문은 2003년 8월 10일에 접수하여 1차 수정을 거쳐 2003년 10월 4일에 게재확정 되었습니다.