실시간 기록을 위한 광매체 API
(A New Optical Media API for Real-Time Recording)

이 민 석 † 송 진 석 †† 윤 찬 희 ††
(Minsuk Lee) (Jinseok Song) (Chanhee Yun)

요 약 현재 시장에는 PVR, DVR, 캠코더 등과 같이 기록 가능 cd, dvd와 같은 광매체에 멀티미디어 스트림을 저장하고, 재생하는 많은 임베디드 시스템들이 많이 있다. 이 논문에서는 다양한 응용과 임베디드 시스템에서 사용할 수 있도록 명확한 구조와 문서를 가진 운영체제 독립적 광매체 API의 설계하고 공개 소스 형태로 구현한 내용을 기술한다. 연구에서는 실시간 광매체 기록을 위하여 ISO-9660 표준을 따르는 새로운 매체 레이아웃과 API를 제안하고 구현하였다. 또 개발된 API의 유용성을 검증하기 위해, 잘 알려진 CD 버닝 프로그램인 *cdrecord*를 대치할 수 있는 텍스트 응용 프로그램과 그래픽 응용 프로그램도 개발하였다. 모든 초기 개발은 Linux PC 환경에서 진행되었으며, 이후 pSOS를 운영체제로 하는 상용 임베디드 시스템에도 이식되었다.

키워드 : 광매체, 실시간 기록, API, 리눅스, 공개소스

Abstract There are many embedded systems which store and play multimedia streams on optical media such as recordable cd and dvd. Some of those are PVRs, DVRs, and camcorders. In this paper we describe the design and implementation of a new, well structured, fully documented, operating system independent and open source optical media API which can be used in various applications and embedded systems. We also design an ISO-9660 compliant optical media layout, an API set and the scenario for real-time recording. To prove the usability, we develop a text application to replace well-known CD-burning software, *cdrecord*, and a graphic burning application. All the implementations are firstly done on Linux PC environment, and then ported to a commercial embedded system which uses pSOS as an operating system.

Key words : optical media, real-time recording, API, Linux, open source

1. 서 론

오늘날 정보통신 분야의 발전은 초고속 네트워크를 기반으로 한 멀티미디어 서비스와 멀티미디어 가전을 중심으로 이루어지고 있다. 특히 보안용 영상 저장 장치인 DVR(digital video recorder), 개인용 방송 녹화 장치인 PVR(personal video recorder), 캠코더 등과 같은 임베디드 시스템들은 멀티미디어 데이타를 실시간으로 저장하고 재생하는 기능을 가지고, 하드 디스크나 고밀도 자기 테이프 등을 저장 매체로 사용하여 왔다. 최근에는 cdr/cdrw, dvd±r/rw와 같은 광매체를 주 저장 매체로 하는 pvr 및 캠코더가 출시되고 있으며, 시장을 점차 넓혀가고 있는 독립형 dvr들도 광매체에 저장되어 있던 영상 데이타를 백업하거나, 실시간으로 기록하기 위한 기능을 추가하고 있다. 실시간 기록이란 이미 존재하는 파일이나 데이타가 아닌 현재 만들어지고 있는 데이타를 실시간으로 광매체에 저장하는 방식으로, 일부 캠코더들이 이 방식으로 사용하고 있으며, 최근에는 dvr이나 의료 영상 장치들도 실시간 광매체 기록을 하고 있다.

본 논문에서는 임베디드 시스템 또는 독립적인 응용 프로그램에서, 광매체에 접근하여 데이타를 읽거나 쓸 수 있는 운영체제 독립적 광매체 접근 API를 공개 소스 형태로 개발한 내용을 기술하였다. 또, 광매체에 멀티미디어 데이타를 실시간으로 저장하기 위한 새로운 광매체 레이아웃을 제안하고, 실시간 기록을 위한 상위 API 또한 설계하고 구현하였다. 논문에서 제안한 API와 API를 검증하기 위한 응용 프로그램들은 커널 버전

· 이 연구는 2005년도 한성대학교 교내 연구비 지원과제임
† 종신회원 : 한성대학교 컴퓨터공학과 교수
minsuk@hansung.ac.kr
†† 학생회원 : 한성대학교 컴퓨터공학과
kopuk@hansung.ac.kr
gutentac@hansung.ac.kr
논문접수 : 2006년 5월 2일
심사완료 : 2007년 3월 4일

2.6을 사용하는 리눅스 PC 환경에서 구현되어 시험된 뒤, 홈페이지를 통해 공개되었으며, 이후 pSOS를 비롯한 다른 실시간 운영체제에도 성공적으로 이식되어 상용 임베디드 시스템에 사용되고 있다.

논문의 구성은 다음과 같다. 2장에서는 광매체 관련 표준과 기존의 광매체 관련 개발 사례에 대하여 살펴보고, 3장에서는 새로운 광매체 API의 필요성을 기술한다. 4장에서는 이 논문에서 구현한 계층별 광매체 API, 실시간 기록을 위한 레이아웃과 API 및 기록 시나리오를 기술하고, 제안된 매체 레이아웃의 효율성을 다른 기록 방식과 비교한다. 마지막으로 5장에서는 논문의 결론을 맺는다.

2. 광매체 표준과 기존 광매체 API 관련 개발 사례

광매체는 읽기 전용 매체(cd, dvd, cdrom, dvd-rom), 일회 기록 매체(cdr, dvd±r), 삭제 및 재기록 가능 매체(cdrw, dvd±rw), 임의 섹터 기록 및 삭제 가능 매체(cdrw, dvd±rw, dvd-ram) 등으로 분류할 수 있다. 읽기 전용 매체는 마스터링을 거쳐 음악, 영화, 데이타를 기록하거나, 그들이 복합되어있는 형태로 배포되어, 사용자는 저장된 콘텐츠를 재생하는 용도로만 사용할 수 있다. 기록 가능 매체는 종류에 따라 단 한 번의 기록 또는 반복적인 기록 및 삭제가 가능한 매체이다. 이 가운데 cd는 매체에 따라 650MB에서 800MB까지(통상 700MB)의 기록 용량을 가지며, dvd는 4.7GB(듀얼 레이어 디스크의 경우 그 두 배)에 해당하는 용량을 가진다. 모든 종류의 광매체에 데이타가 저장되는 물리적/논리적 방식은 호환성을 위하여 표준으로 확립되어 왔다[1-3].

2.1 광매체 상의 데이타 저장 구조와 기록 방식

모든 광매체 상의 데이타는 표준에 의해 구체적인 물리적, 논리적 저장 구조가 결정되어 있다. 이 절에서는 대표적인 광 저장 매체인 cd의 데이타 저장 구조와 주요 기록 방식을 간단하게 설명하고자 한다.

하나의 cd는 그림 1과 같이 PCA, PMA 영역과 하나 이상의 세션으로 구성된다. 각 세션은 다시 리드-인 영역과 하나 이상의 데이타 트랙을 갖는 데이타 영역, 리드-아웃 영역으로 구성된다. 대부분의 데이타 cdrom의 경우 하나의 세션으로 구성되며, 여러 개의 세션이 있는 cdrom이 드라이브에 삽입되는 경우에도 대부분 운영체제는 마지막 세션에 있는 메타 데이타만을 읽게 된다.

PCA(power calibration area) 영역은 cd의 가장 안쪽에 위치하는 물리적 공간으로, 드라이브 하드웨어가데이타를 기록할 때, 매체의 특성에 따라 적절한 레이저 빔의 세기를 찾기 위해서 사용하는 영역이다. PMA

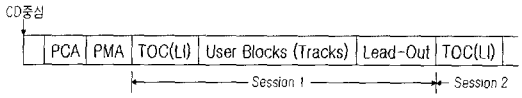

그림 1 CD의 레이아웃

(program memory area) 영역은 트랙의 수, 트랙의 시작 부분과 끝 부분에 대한 정보를 임시로 저장하는 공간으로서 cd의 기록이 완료되기 전까지의 트랙에 관한 메타 데이타를 기록한다. TOC(table of contents) 영역은 리드-인 영역이라고도 불리며, cd에 저장된 데이타에 대한 메타 정보를 기록한 곳으로서 데이타 영역의 전체 길이, 데이타 시작 위치, 디스크 이름, 트랙 수 등의 정보를 가지고 있다. 이 영역은 cd 기록이 완료된 후, pma 영역에 임시로 기록되었던 정보를 바탕으로 기록된다. 첫 번째 toc 영역은 사용자에게는 보이지 않는 영역으로 섹터 주소(LBA, logical block address)를 기준으로 할 때 최초의 논리적 접근이 가능한 주소인 0번 이하의 주소를 가지고 있다. 데이타 영역인 사용자 블록 영역은 실제 사용자가 기록하려는 데이타가 저장되는 영역이다. 사용자 블록 영역은 하나 이상의 트랙으로 구성된다. 오디오 cd의 경우 하나의 트랙이 노래 한 곡에 해당하며, cdrom의 경우 하나의 트랙이 한 개 이상의 파일을 가진 ISO-9660 파일 시스템[4]에 해당된다. 데이타 영역의 각 트랙은 다시 연속적인 데이타 섹터로 구성된다. 마지막으로 세션의 제일 뒤쪽을 차지하고 있는 리드-아웃 영역은 한 세션이 끝났음을 알리는 영역으로 데이타가 없는 빈 공간이다.

광매체에 대한 데이타 기록 방식에는 DAO(disk at once), SAO(session at once), TAO(track at once) 등 세 가지 모드가 있다. 우선 DAO 모드의 경우 매체의 처음 위치부터 중단 없이 전체 데이타를 기록하는 방식으로 하나의 세션으로 디스크가 구성된다.

TAO 모드는 데이타 기록 단위를 트랙 단위로 하는 모드이다. DAO 모드와는 달리 리드-인 영역의 데이타를 명시할 필요가 없으며, 기록 완료 명령이 내려지면, 드라이브가 자동으로 리드-인 영역에 매체 내의 트랙들에 관한 정보를 기록한다. 한 트랙의 최소 크기는 300 섹터이며, 각 트랙과 트랙 사이에는 150 섹터의 빈 공간이 존재한다. 트랙 단위로 기록을 하게 될 경우 기록 완료 명령을 내리기 전까지 최고 99트랙까지 연속하여 기록이 가능하다. 하지만 기록 완료 명령에 의해 리드-인 영역이 유효한 정보로 채워지기 전까지는 운영체제가 하나의 세션의 완료로 판단하지 않아 유효한 매체로 인식하지 못한다.

SAO 모드는 기록 단위를 운영체제에서 인식 가능한 세션 단위로 하되, 매체의 재활용성을 높인 기록 모드라

고 할 수 있다. SAO 모드로 여러 차례 기록된 미디어를 멀티 세션 디스크라고 한다. 멀티 세션으로 기록된 매체는 각 세션마다 각각 약 9MB와 13.5MB 크기의 리드-인, 리드-아웃 영역을 포함하기 때문에 할당하기 때문에 실제 사용자 데이타를 위한 저장 공간은 단일 세션 매체보다 적다.

2.2 ISO-9660 파일 시스템

ISO-9660 파일 시스템[4]은 cdrom의 기본 파일 시스템으로 광매체에서 가장 많이 사용되는 파일 시스템이며, 운영체제 독립적이기 때문에 대부분의 운영체제에서 인식이 가능하다. dvd-rom의 경우 udf 파일 시스템[5] 사용을 권장하고 있지만 시장에 나와 있는 많은 기록 프로그램들은 cdrom처럼 ISO-9660 파일 시스템을 기본으로 지원하고 있다. 이 절에서는 본 논문이 실시간 기록을 레이아웃 설계에서 응용하고 있는 방식인 ISO-9660 파일 시스템을 간단히 기술한다.

ISO-9660 파일 시스템은 다른 모든 파일 시스템과 마찬가지로 데이타 파일들과 메타 데이타로 구성되며, 메타 데이타는 PVD(primary volume descriptor), 경로 테이블, 디렉토리 구조 세 가지로 구성된다. 그들 사이의 관계는 그림 2와 같다.

PVD는 언제나 각 ISO-9660 세션 트랙의 16번째 lba에 존재하는 파일 시스템의 최상위 메타 데이타로서 경로 테이블 및 디렉토리 구조의 위치 정보, 파일 시스템의 데이타 크기, 매체 속성 식별자 등의 데이타를 지니고 있다. 경로 테이블은 ISO-9660 파일 시스템 내의 디렉토리에 대한 일종의 인덱스로서 각 디렉토리의 이름과 함께 디렉토리 내부 정보의 시작 위치를 지니고 있어 특정 디렉토리의 내용에 바로 접근할 수 있도록 해준다. 디렉토리 구조는 실제 ISO-9660 파일 시스템에 기록된 파일의 이름과 날짜 및 시간, 파일 시작 위치, 디렉토리 등 파일 시스템 내의 모든 파일과 디렉토리에 관한 모든 메타 데이타를 가지고 있다.

각 트랙이 ISO-9660 파일 시스템으로 구성된 멀티세션 디스크의 경우, 디렉토리 구조 내의 파일 정보에 이전 트랙들에 저장된, 즉 먼저 기록된 세션 내의, 파일들의 위치도 지정할 수 있으며 이를 이용하여 cd를 점진적 백업 용도로 이용할 수 있다.

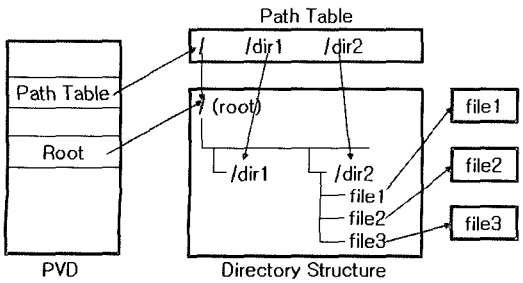

그림 2 ISO-9660 파일 시스템 구조

2.3 기존 광매체 관련 개발 사례

광매체 드라이브에 접근하여 매체의 데이타를 읽거나 매체에 데이타를 기록하는 방식은 시스템과 광매체 드라이브 인터페이스에 관한 표준과 광매체 접근에 관한 표준으로 잘 정의되어왔다. 특히 광매체 접근에 관한 표준은 국제 기술 위원회 T10에 의하여 표준으로 개발된 MMC(multi-media commands)로서, 모든 종류의 cd/dvd 광매체 구동 하드웨어를 제어하기 위한 명령들을 정의하고 있다[6]. 이러한 표준들을 바탕으로 시장에서는 많은 운영체제를 위하여 다양한 광매체 접근을 위한 API와 응용 프로그램을 개발하여 왔다.

먼저 API 관련 사례를 살펴보면, 많은 저수준 API들은 광매체 만을 대상으로 하는 것이 아니고, 광 드라이브들이 주로 사용하는 인터페이스 방법인 scsi[7], atapi[8]에 대한 API들이다. 우선 마이크로소프트사의 윈도우즈 계열 운영체제에는 aspi[9]가 있으며, 윈도우즈 ddk가 제공하는 scsi 연결 기능을 이용할 수도 있다. 리눅스 운영체제에서는 같은 기능을 디바이스 드라이버 수준에서 제공하고 있다. 리눅스 커널 2.4까지는 scsi 드라이버 또는 ide 드라이버 위에서 패킷 명령 프로토콜을 지원하는 scsi 에뮬레이터를 이용하여 광매체에 접근할 수 있으며, 리눅스 커널 2.6에서는 ide 드라이버에서 직접 패킷 명령 프로토콜을 지원한다.

저수준인 mmc 명령을 바탕으로 트랙, 세션 수준의 API가 별도로 개발되어 발표된 사례는 찾아볼 수 없으며 응용 프로그램의 일부로만 개발되어 이용되고 있다. 윈도우즈와 리눅스 등 거의 모든 운영체제에서 실행 가능한 *cdrecord*[10]는 이 수준의 API를 명령어 라인 인터페이스로 제공하고 있다.

최상위 계층이라고 할 수 있는 레이아웃 수준의 API는 개발 사례가 많이 있다. ISO-9660, udf의 경우[11] 등에서 상업적으로 개발된 API를 찾을 수 있으며, 공개 소스 형태로는 명령어 라인 수준의 인터페이스만을 제공하는 *mkisofs*[12] 등이 있다.

일반 사용자를 대상으로 하는 응용 프로그램 수준의 구현 사례는, 그 상업적 가치 때문에, 모두 나열하기가 어려울 정도로 많다. [13] 등 여러 회사에서 만들어진 기록 소프트웨어들은 대부분 윈도우즈 운영체제를 위하여 것들이다. 특히 마이크로소프트사의 윈도우즈 XP에는 여러 형식의 cd/dvd를 만들어 낼 수 있는 기능이 자체적으로 포함되어 있기도 하다. 공개 소스 진영에는 *cdrecord*[10]와 같은 텍스트 기반 cd/dvd 기록 프로그램을 바탕으로 *cdrecord*의 GUI에 해당하는 그래픽 전처리기 형태로 구현된 [14] 등이 있다.

3. 새로운 광매체 API의 필요성

이 장에서는 기존의 광매체 관련 공개 소스 개발 결과 가운데 가장 널리 사용되고 있는 *cdrecord*를 기준으로 기존 광매체 관련 개발의 문제점을 살펴보고 새로운 공개 소스 광매체 API의 필요성과 특히 실시간 광매체 기록 API 개발의 필요성을 기술한다.

3.1 기존 광매체 관련 개발의 문제점

이 절에서는 특히 리눅스, 윈도우즈 등 거의 모든 운영체제에 이식되어 현재 가장 많이 사용되고 있고, 안정적인 공개 소스 광매체 기록 프로그램인 *cdrecord*가 가진 문제점에 대하여 살펴본다. *cdrecord*는 일반 응용 프로그램과 임베디드 멀티미디어 시스템에서 현실적으로 이용 가능한 유일한 공개 소스 광매체 프로그램이기 때문에, 공개 소스 형태의 광매체 API 개발을 추구하는 이 논문에서 *cdrecord*의 문제점을 살펴보는 것은 매우 중요하다.

*cdrecord*는 명령어 라인 인터페이스만을 제공하는 독립적인 응용 프로그램으로서, 응용 프로그램과 *cdrecord* 둘 사이의 데이타 전달 과정에서, 메모리 등 시스템 자원에 대한 부수적인 요구가 따른다. 또 *cdrecord*는 cd/dvd에 여러 가지 형식으로 데이타를 읽고 쓰는 가장 중요한 부분만 소스가 25,000 라인 정도에 달하여 소스 관리가 어렵고, 실행 파일이 크다. 그리고 동작 구조 상 유닉스 스타일의 시스템에 의존하도록 설계되었기 때문에, POSIX 스타일의 장치 인터페이스 및 프로세스 간 통신 인터페이스가 있는 경우에만 이식이 가능하다. 그리고 소스 코드 관점에서는 표준이 잘 정립되지 않았던 초창기의 다양한 광매체 드라이브들을 지원해 오면서 소프트웨어의 계층 구조가 흐트러진 상태가 되었다. 마지막으로 내부 소프트웨어 구조, 구현 내용, 내부 라이브러리에 대한 문서가 존재하지 않는다. 결론적으로 *cdrecord*는 응용 프로그램 자체는 매우 유용하다고 볼 수 있지만, 임베디드 시스템과 같은 다른 응용에서 활용하기 위해서는 많은 소스를 읽고 이해하고, 수정해야 하는 어려움이 따른다.

따라서 임베디드 시스템 개발자와 독립적인 응용 프로그램 개발자들이 자신의 응용 프로그램에서 직접 광매체 드라이브를 다양한 수준에서 제어하여 cd/dvd에 데이타를 기록할 수 있는 공개 소스 형태의 새로운 광매체 접근 API가 필요하다. 구체적으로는 광매체 기록의 의미 있는 단위인 트랙, 세션 수준에서 추상화된 명령 체계, 오류 복구 등을 지원하는 API가 필요하며, 임베디드 멀티미디어 시스템 업계에서 요구하고 있는 실시간 기록을 위한 레이아웃 API도 역시 필요하다.

3.2 실시간 기록 API의 필요성

실시간 기록이란 현재 생성되고 있는 데이타를 실시간으로 기록하고, 모든 데이타에 대한 기록이 끝나면 기록된 데이타에 관한 정보(즉, 파일 크기 등과 같은 메타 데이타)를 기록하여 나중에 읽을 수 있도록 하는 기록 방식을 의미한다.

대부분의 광매체 기록 프로그램들은 매체에 기록되어야 하는 파일들을 가지고 먼저 이미지 파일(매체에 기록될 ISO-9660 또는 udf 파일 시스템 전체를 하나의 파일로 만든 것)을 만든 후 그 파일을 dao 모드로 기록하는 방법을 사용한다. 즉, 기록할 모든 데이타가 사전에 준비되어야 하고, 또 이미지 파일을 임시로 저장할 공간이 필요하게 되어, 실시간으로 데이타를 저장해야하는 응용이나, 수백 MB 또는 수 GB에 달하는 원본 데이타 파일과 이미지 파일을 저장할 공간이 없는 소형 시스템이나 임베디드 시스템에서는 이용이 불가능하게 된다.[1)]

실시간 기록을 쉽게 구현하지 못하는 가장 중요한 이유는 광매체의 물리적인 특성 상 처음부터 순차적인 데이타 기록만 가능할 뿐 임의의 위치에 대한 기록이 불가능하기 때문이다. 즉, ISO-9660 파일 시스템과 같이 저장된 파일들에 관한 정보가 매체의 앞부분에 존재하는 경우, 데이타의 크기와 같이 사전에 값을 알 수 없는 값을, 데이타를 먼저 기록한 뒤 나중에 메타 데이타 영역에 기록할 수 없다. 또 기존 프로그램을 이용하여 데이타를 기록하고 매체에 대한 최종적인 기록 완료 명령을 내리기 전에 매체를 드라이브에서 꺼내거나, 전원을 껐다가 나중에 그 매체에 추가 기록하거나, 다른 드라이브에서 기록되던 매체에 추가 기록하는 것을 허용하는 것이 쉽지 않다. 이 과정에서의 문제는 매체를 드라이브에서 꺼내거나 전원이 차단되는 시점까지의 기록한 데이타(파일)에 대한 메타 데이타를 매체 자체에 보존하고 나중에 읽어낼 방법이 어떤 레이아웃 표준에서도 기술되어있지 않다는 점이다.

기술적으로는 하나의 파일이 하나의 ISO-9660 파일 시스템을 구성하도록 하여 멀티 세션 매체를 만드는 방법 또는 cd-rw/dvd±rw 또는 dvd-ram을 이용한 패킷 모드 기록 방식을 사용하여 제한적인 실시간 기록을 할 수 있다. 멀티 세션 방식의 경우 저장할 데이타의 크기를 사전에 알 수 있는 경우에만 ISO-9660 메타 데이타를 구성할 수 있어 여러 실시간 기록 응용에 적합하지

1) 실제로 많은 광매체 기록 프로그램들은 이미지 파일을 디스크에 생성하지 않고, *pipe*와 같은 프로세스 간 통신 기법을 이용하여 파일 시스템 이미지를 광매체 기록 프로그램에 전달하는 것이 보통이다. 하지만 이 경우에도 이미지 생성에 필요한 원본 데이타 파일은 어딘가에 저장되어 있어야 하므로 그를 위한 공간이 필요하다.

않다. 또 세션의 경계를 만들고 새로운 세션을 시작하는 시간이 길어 연속적으로 여러 파일을 기록하는데 상당한 시간적 오버헤드가 존재하고, 세션을 유지하기 위한 저장 용량의 오버헤드도 크다. 패킷 모드 기록 방식의 경우 데이타 기록 성능이 매우 낮고 저장 매체의 단위 비용이 크게 상승하며, 패킷 모드로 기록된 매체를 읽을 수 있는 장치가 많이 보급되지 않아 호환성이 부족하다는 문제가 있다.

이 때문에 dvr, pvr이나 캠코더와 같은 실시간 영상 저장 장치들이 이미지 파일을 만들지 않으면서, 또 저장되어야 할 데이타의 크기와 같은 여러 정보를 사전에 알지 못한다 해도 점진적으로 광매체에 발생하는 데이타를 기록할 수 있는, 실시간 기록을 위한 효율적인 광매체 레이아웃과 그를 지원하는 API가 필요하다.

4. 새로운 광매체 API의 구현

이 장에서는 이 연구에서 개발한 새로운 광매체 API의 개발 목표를 제시하고, 개발 목표에 따른 구현 결과, 각 계층별 구현의 세부 내용을 기술한다. 특히 실시간 기록 API의 경우, 본 연구에서 제안한 새로운 광매체 레이아웃과, 제안된 레이아웃에 따른 기록 시나리오를 설명하고 및 레이아웃의 효율성을 검증한다.

4.1 새로운 광매체 API의 개발 목표

본 논문의 연구에서 새로운 광매체 API를 개발함에 있어 설정된 목표들은 다음과 같다.

- 표준에 입각한 계층 구조 : 광매체의 레이아웃, 기록 방식, 광 드라이브에 관한 접근은 모두 표준으로 정의된다. 개발된 API는 각 표준의 개정 및 새로운 광매체 및 기록 방식의 추가에 원활히 대응하기 위하여 표준에 입각한 계층 구조를 유지한다.
- 이식 가능한 API : 개발된 광매체 API는 시스템 종속적인 요소를 제거하고, 운영체제 적응 계층을 두어 광 드라이브에 접근함으로써 다양한 운영체제 환경과 운영체제가 없을 수도 있는 임베디드 시스템에도 이식되어 사용 가능하다.
- 호환성 있는 실시간 기록 지원 : 응용 프로그램에 실시간 데이타 기록 기능을 제공하되, 기록된 매체의 레이아웃이 표준을 벗어나지 않도록 하여 호환성을 유지한다. 개발된 실시간 기록 레이아웃은 ISO-9660 표준을 따름으로써 호환성을 확보하여 광매체를 지원하는 대부분의 장치와 운영체제에서 바로 인식된다.
- 공개 소스로서의 활용성 제고 : 개발된 API는 공개 소스 라이선스를 따르며, 홈페이지를 통해 공개된다. 공개 소스의 API로서, 여러 응용 프로그램과 임베디드 시스템 등에 널리 이용될 수 있도록 모든 이용 가능한 API, 활용 시나리오, 내부 구조 등을 문서화한다.
- 기존 프로그램과의 호환성 : 기존 광매체 기록 응용 프로그램들과 GUI 전처리 프로그램들을 본 연구에서 개발된 API 상에서도 이용할 수 있도록, API와 더불어 cdrecord와 호환성 있는 명령어 라인 인터페이스를 가진 응용 프로그램도 함께 제공한다.

4.2 API의 전체 구조와 구현 결과

구현된 API의 계층 구조는 그림 3과 같다. 그림에서 보는 바와 같이 API는 mmc 표준에 정의되어있는 패킷 명령과 1:1로 대응되는 mmc API와 그 mmc API를 바탕으로 각종 기록 모드에 따른 쓰기 동작이나, 드라이브의 제어의 추상화 등을 구현한 유틸리티 API, 그리고 운영체제 입장에서 보는 매체의 논리적 형식이나 파일 시스템과 같이 광매체의 레이아웃을 담당하는 레이아웃 API로 구성된다. mmc API 하부에는 운영체제 적응 계층이 있다.

현재 mmc API는 mmc 버전 4를 기준으로 구현되어 있으며, 유틸리티 API는 광매체가 지원하는 다양한 쓰기 모드를 모두 지원할 수 있도록 구현되었고, 레이아웃 API는 ISO-9660 파일 시스템 지원 모듈과 ISO-9660 지원 모듈을 이용한 실시간 기록 모듈까지 구현되었다. 또 구현된 API를 시험하고 *cdrecord*를 대치하여 기존에 존재하던 GUI 전처리기 프로그램들을 그대로 이용할 수 있도록 하는 명령어 수준 인터페이스를 제공하는 텍스트 기반 기록 프로그램과, 구현된 API를 응용 프로그램에서 직접 이용하는 그래픽 기록 프로그램도 공개 소스인 *GnomeBaker*[14]를 수정하여 개발하였다.

모든 API와 응용 프로그램 초기 구현은 리눅스 커널 버전 2.6을 사용하는 리눅스 PC 상에서 atapi 인터페이스를 가지는 cd/dvd 드라이브를 대상으로 이루어졌다. 또, 운영 체제 적응 계층에 대한 약간의 수정을 거쳐 실

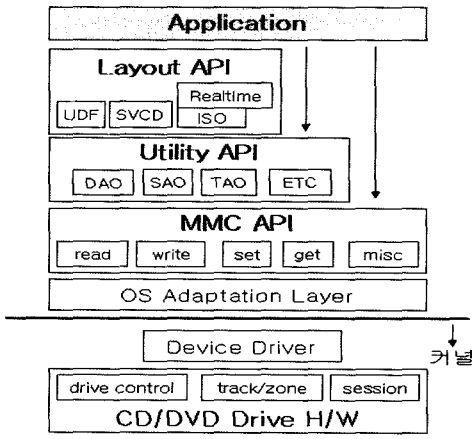

그림 3 구현된 광매체 API의 구조

시간 운영체제인 pSOS와 같은 실시간 운영체제에도 성공적으로 이식되었으며, usb 인터페이스를 갖는 광매체 드라이브 등 다양한 드라이브 인터페이스에서도 문제없이 동작함을 확인하였다.

4.2.1 운영체제 적응 계층

운영체제 적응 계층은 다양한 운영체제 또는 운영체제가 없는 환경에서 광 매체 드라이브의 사용 권한을 얻고, mmc 계층으로부터의 명령을 디바이스 드라이버에 전달한 뒤, 결과를 다시 mmc 계층에 전달하는 과정에서 인터페이스 변환을 담당하는 매우 단순한 역할을 한다. 즉, 이 계층은 *cd_open(), cd_close(), cd_ioctl()* 3개의 API만을 가지고 각 운영체제가 제공하는 장치 드라이버에 접근한다. 따라서 운영체제가 없는 환경에서도 광 드라이브 접근을 위한 이 세 가지 API 만을 구현함으로써 이 논문에서 개발된 광매체 API를 이용할 수 있다.

실제 *cd_ioctl()* 함수에 의해 전달되는 제어 명령은 리눅스 커널 2.6의 ide-cd 드라이버에 사용된 것과 같은 것으로 표 1의 *cdrom_generic_command* 구조체이다.

실제 mmc 표준에서 정의된 명령 자체는 표 1의 구조에서 *cmd* 배열을 통해 하드웨어에 전달되며, *buffer*는 지정된 명령에 사용되는 데이타의 포인터이다. 또 각 명령에 대한 처리 결과와 에러의 내용은 *request_sense* 구조를 통하여 알 수 있으며, 상위 계층에서는 모든 명령에 대하여, 리턴된 *sense*의 각 필드의 값을 확인하고, 에러에 대처한다.

표 1 광매체 제어 명령

```
struct cdrom_generic_command {
  char cmd[cdROM_PACKET_SIZE]; // MMC 명령 패킷
  char *buffer;                // 명령에 따른 데이타
  unsigned buflen;             // 데이타 크기
  struct request_sense *sense; // 명령 처리 결과
    ...
```

4.2.2 MMC API 계층

표준에 따른 각 mmc 명령은 표 2와 같이 12 바이트 크기의 구조체로 구성된다. 원래 mmc는 scsi 인터페이스를 가지는 광매체 드라이브를 위해 개발되었으며, 현재 대부분의 광매체 드라이브가 사용하는 atapi 인터페이스에서는 패킷 명령 프로토콜을 이용하여 같은 mmc 명령을 드라이브에 전달하는 방식으로 동작한다.

표 2 MMC 명령 패킷 구성

	0	1	2	3	4	5	6	7
0	Operation code (Command)							
1	Reserved			Control field or parameter				
2-5	Logical block address							
6-9	Transfer length or parameters							
10	Reserved							
11	Control field							

구현된 mmc API의 주요 함수와 그 기능은 표 3과 같다. mmc API의 경우 mmc 표준에 정의되어 있는 각 명령이 하나의 API 함수에 대응된다.

가장 기본적인 함수인 *cdrwmmc_read_block()*과 *cdrwmmc_write_block()*은 섹터 단위의 읽기 및 쓰기를 지원한다. 드라이브에 삽입된 cd/dvd 매체와 각 트랙에 대한 정보를 얻기 위한 API로는 *cdrwmmc_read_disc_info()*와 *cdrwmmc_ read_track_info()*가 있다. 현재 기록 가능한 섹터에서부터 지정하는 크기의 공간을, 나중에 기록할 목적으로, 예약하여 비워두기 위한 *cdrwmmc_reserve_track()*도 mmc API 가운데 하나이다. *cdrwmmc_start_stop()*은 전달하는 인자 값에 따라 트레이를 열고 닫거나 드라이브의 상태를 전환한다. 실제 데이타를 cd/dvd 매체에 기록하기 전에 기록 모드, 섹터 크기 등의 정보를 보내는 API는 *cdrwmmc_mode_select()* 함수이다. 이 API는 쓰기 파라메터 페이지 구조를 이용하여 기록 모드, 섹터 구조 등 필요한 정보를

표 3 주요 MMC API와 기능

API	기능
cdrwmmc_read_block	매체의 데이타를 섹터 단위로 읽기
cdrwmmc_read_disc_info	매체의 상태 정보 읽기
cdrwmmc_read_track_info	트랙의 상태 정보 읽기
cdrwmmc_write_block	섹터 단위로 매체에 쓰기
cdrwmmc_close_track_session	매체의 트랙/세션 쓰기 종료
cdrwmmc_mode_select	드라이브에 쓰기 모드 설정
cdrwmmc_mode_sense	현재 드라이브에 설정된 쓰기 모드 읽기
cdrwmmc_prevent_medium	드라이브의 트레이 상태의 잠금 상태 변경
cdrwmmc_reserve_track	매체에 일정 크기 섹터를 한 트랙으로 예약
cdrwmmc_start_stop	드라이브의 트레이 열고 닫기
cdrwmmc_test_unit_ready	드라이브 상태 검사

먼저 설정하게 된다. 이 명령 이후에 호출되는 *cdrwmmc_write_block()* 함수에 의한 데이타 쓰기는 모두 같은 모드로 이루어진다.

4.2.3 유틸리티 API 계층

표 4에는 유틸리티 API 계층에 속한 주요 함수들이 나열되어 있다. 유틸리티 API 계층은 여러 mmc API를 조합하거나 추상화하여 개별 기능이 아닌, 매체로의, 의미 있는 접근 방법을 제공한다. 즉 mmc 명령의 복잡한 인자 구조를 추상화하고 복구 가능한 에러에 대한 처리를 내부적으로 수행하여 응용 프로그램 또는 상위 계층 API에 이해하기 쉽고, 신뢰성 있는 매체 접근 방법을 제공한다. 또 대부분의 광매체 기록을 위한 응용 프로그램에게 의미 있는 데이타 저장 단위인 트랙, 세션을 제어하기 위한 API들을 포함한다.

4.3 실시간 광매체 기록을 위한 API

4.3.1 실시간 기록을 위한 매체 레이아웃

본 논문에서는 실시간 기록에 필요한 매체 레이아웃을 새로 구성하였다. 제안된 레이아웃은 ISO-9660 파일 시스템 표준을 준수하고 있어, 이 방식으로 기록된 매체는 모든 범용 운영체제에서 정상적인 매체로 인식되어 저장된 데이타를 읽을 수 있다. 구성된 레이아웃은 그림 4와 같은 구조를 가진다.

실시간 기록을 위한 레이아웃에는 보통의 ISO-9660 형식의 매체가 하나의 데이타 트랙으로 구성된 것과는 달리 멀티 트랙으로 구성된다. 그림 4에서 *n*은 트랙의 개수로 표준에서 제한하고 있는 99개를 최댓값으로 한다. 각 트랙에는 하나의 단위 데이타(또는 파일)과 현재 트랙까지 기록된 데이타들에 관한 정보가 TMP Data로 표시된 중간 정보 영역에 저장된다. 각 트랙의 중간 정보 영역은 트랙 2부터 해당 트랙의 데이타에 이어 저장되며, 그 내용은 트랙 2부터 해당 트랙까지 기록된 파일들에 대한 ISO-9660 파일 시스템의 메타 데이타이다. 따라서 모든 데이타가 트랙 별로 기록된 후, 최종적으로 확정된 ISO-9660 메타 데이타는 매체의 마지막 트랙의 중간 정보 영역에 기록된 데이타와 같다. 중간 정보 영역에 저장된 데이타의 양은 최대로 98개의 파일이 있다 해도 수 KB를 넘지 않는 크기로 오버헤드가 적다. 또 많은 실시간 기록 응용이 영상 데이타 응용임으로 감안할 때, 98개의 파일 수 제한은 문제가 되지 않을 것으로 기대된다.

트랙 1은 모든 데이타에 대한 쓰기가 완료된 후 매체 전체에 대한 ISO-9660 메타 데이타가 저장되는 공간으로, 실시간 데이타 기록을 시작하기 전에 *cdrwmmc_reserve_track()*을 이용하여 예약된다. 실제 ISO-9660 파일 시스템 메타 데이타를 위해 필요한 공간은 중간 정보 영역과 같은 수 KB에 불과하지만 표준에서 제시하고 있는 최소 트랙 크기인 300 섹터를 차지한다. 이 영역은 모든 사용자 데이타가 기록된 후 기록 완료 명

표 4 주요 유틸리티 API와 기능

API	기능
cdrwutil_data_write	임의 크기의 데이타 매체에 쓰기
cdrwutil_track_mode_select	트랙의 쓰기 모드 설정
cdrwutil_close_track	트랙에 대한 쓰기 완료 명령
cdrwutil_close_medium	매체에 대한 쓰기 완료 (finalize) 명령
cdrwutil_last_writable	매체에 쓰기가 가능한 섹터의 시작 주소를 반환
cdrwutil_close_track_session	매체의 트랙/세션 쓰기 종료
cdrwutil_tray_open	트레이 열기
cdrwutil_tray_close	트레이 닫기
cdrwutil_tray_lock	트레이 잠금
cdrwutil_tray_unlock	트레이 잠금 해제
cdrwutil_device_check	드라이버/매체 상태 확인
cdrwutil_check_cd	매체에의 쓰기 가능 여부 확인
cdrwutil_number_of_track	현재까지 기록된 트랙 수 확인
cdrwutil_remain_check	매체에 기록 가능한 남은 용량 확인

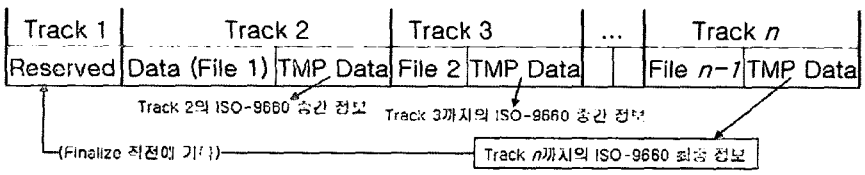

그림 4 실시간 기록을 위한 레이아웃

령 직전에 마지막 트랙의 중간 정보를 그대로 복사하여 기록한다. 결과적으로 기록이 완료된 매체의 트랙 1에는 실제 사용자 데이타는 없고, 사용자 데이타의 위치에 대한 파일 시스템 메타 데이타만 존재하며 그 메타 데이타에서 각 파일의 위치는 각기 다른 트랙의 사용자 데이타 시작 위치를 가리키게 된다.

4.3.2 실시간 기록을 위한 API와 기록 시나리오

표 5에는 실시간 기록을 위해 설계된 주요 API와 그 기능을 기술하였다.

구현된 API를 이용하여 광매체에 실시간 기록을 하는 과정은 그림 5의 시나리오에서 보이고 있다. 1단계에서는 광매체 드라이브를 초기화하고, 디바이스에 관련된 전반적인 설정을 한다.

2단계에서는 현재 드라이브에 삽입된 매체의 상태(한 번도 기록이 없었던 빈 매체, 이미 이 논문에서 제안하는 방식으로 기록이 진행 중인 매체, 기록이 완료되어 더 이상 쓸 수 없는 매체 등)를 확인하며, 이 방식으로 기록이 진행 중인 매체라면 현재까지 기록된 트랙의 수와 남은 용량을 확인한다.

3단계는 자원 및 데이타 초기화 단계로, 중간 정보에 필요한 메모리를 할당받아 초기화하며, 빈 매체가 삽입된 경우, 최종적인 ISO-9660 파일 시스템 메타 데이타 저장을 위하여 첫 번째 트랙을 예약한다.

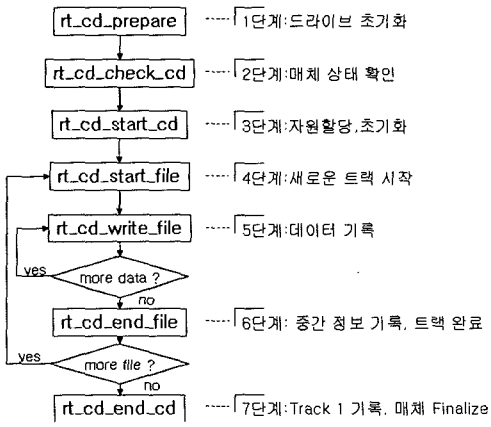

그림 5 실시간 기록 시나리오

4단계는 새로 저장할 데이타에 대한 설정을 하는 단계로서, 디렉토리 및 파일 이름 정보를 입력 받아 (실시간 기록의 경우, 기록할 데이타의 크기는 이때 알지 못한다) 매체에 기록될 사용자 데이타에 대한 메타 정보를 설정한다. 또, 이 시나리오에 따라 실시간 기록한 트랙이 존재하는 매체가 드라이브에 새로 삽입된 경우라면, 마지막 트랙의 중간 정보 영역을 3단계에서 할당받은 메모리에 읽는다.

5단계는 실제 사용자 데이타를 기록하는 단계로서 한 트랙에 해당하는 데이타 파일을 기록하게 된다. 한 트랙에 기록할 수 있는 데이타의 최대 크기는 매체의 남은 용량이 된다.

6단계에서는 하나의 단위 데이타에 대한 쓰기를 완료하여 기록된 데이타의 크기를 확정하고, 현재 트랙까지의 중간 정보를 같은 트랙의 뒷부분에 기록한다. 즉 단계 5에서 트랙에 사용자 데이타를 매체에 기록한 후, 트랙 완료 명령을 내리지 않은 상태에서, 마지막 사용자 데이타 섹터의 다음 섹터에서부터 중간 정보를 기록하게 된다. 이로서 하나의 트랙은 사용자 데이타와 중간 정보의 연속된 형태가 되지만, 최종적인 ISO-9660의 메타 데이타에는 응용 프로그램이 사용자 데이타 부분만 읽을 수 있도록 각 파일의 크기가 지정되어 기록된다. 이 단계까지 완료된 뒤에는 지금까지 기록한 파일들의 이름, 크기, 기록 날짜 등의 데이타가 매체에 중간 정보로 남아 매체를 드라이브에서 꺼낼 수도 있고, 나중에 추가해서 기록하는 것도 가능하다. 하지만 이 상태의 매체는 기록 완료 과정을 거치지 않았기 때문에 범용 운영체제에서는 정상적인 매체로 인식되지는 않는다.

7단계는 매체에 대한 기록 완료 단계이다. 이 단계에서는 최종적인 ISO-9660 파일 시스템 메타 데이타를, 앞서 단계 3에서 예약된 1번 트랙에 기록을 하고 매체 기록 완료 명령을 내린 후, 할당받은 모든 자원을 반환한다. 이 단계까지 완료된 매체는 ISO-9660 파일 시스템이 지원되는 모든 운영체제에서 성공적으로 인식되어 데이타 읽기가 가능하다.

4.4 제안된 레이아웃의 효율성 평가

모든 광매체 기록 소프트웨어는 데이타를 준비하고,

표 5 실시간 기록 API와 기능

API	기능
rt_cd_prepare	실시간 기록을 위한 기록 매체 준비
rt_cd_check_cd	매체 상태, 남은 용량 체크
rt_cd_start_cd	데이타 구조 초기화
rt_cd_start_file	새로운 파일에 대한 데이타 구조 설정, 트랙 기록 초기화
rt_cd_write_file	매체에 데이타 쓰기
rt_cd_end_file	TMP Data 기록 및 트랙 쓰기 완료
rt_cd_end_cd	트랙 1 쓰기 및 매체 쓰기 완료 (finalize)

광매체 드라이브에 명령과 그 명령에 따른 데이타를 인터페이스가 지원하는 방식으로 전달하고, 명령 처리 결과를 기다리는 과정을 반복한다. 실제 광매체 기록 성능은 API보다는 광매체 드라이브의 하드웨어 성능, 인터페이스 종류에 의해 결정된다. 그 가운데 물리적인 광 드라이브 하드웨어의 배속 수(표준 속도 대비 몇 배의 속도로 읽기, 또는 쓰기를 할 수 있는지)가 성능을 결정하는 가장 결정적인 요인이며, 광 드라이브 내부의 버퍼 크기, 그리고 ide, scsi 등 광 드라이브와의 인터페이스가 제공하는 데이타 전송 속도가 미미하지만 그 다음의 요인들이 된다. 이 논문의 구현 대상인 API는 응용 프로그램이 준비한 데이타를 광 드라이브에 전달하고, 그 결과를 다시 응용 프로그램에 보고하는 중간적인 역할을 하는 소프트웨어 계층으로서 성능 평가 인자로서 실행 시간은 적합하지 않은 것으로 판단된다. 이 논문에서는 보통의 cd-r, dvd-r과 같은 매체에, 실시간 기록과 가장 근접한 기록 방법을 제공할 수 있는 멀티 세션 방식과 저장 공간 이용의 효율성을 비교하고자 한다.

멀티 세션 기록 방식은, 메타 정보를 사전에 알아야 하는 제약 때문에, 생성되고 있는 데이타에 대한 실시간 기록이 불가능하여, 각 기록 대상 데이타를 포함하는 ISO-9660 이미지를 생성한 뒤, 하나의 세션으로 기록하는 방식이다. 두 번째 세션부터는 앞쪽 세션에 있는 데이타에 관한 메타 데이타를 포함함으로써 이전에 다른 세션으로 기록된 데이타에 접근할 수 있다. 따라서 멀티 세션 기록 방식은 데이타 또는 파일 시스템 이미지를 임시로 저장할 별도의 메모리 또는 디스크 공간이 필요하며, 저장된 데이타를 다시 읽어 광매체에 기록함으로써 같은 데이타를 기록할 때, 소요되는 시간이 이 논문의 실시간 기록 방식과는 비교할 수 없을 정도로 길다는 것은 명백하다. 따라서 이 논문에서는 실시간 기록에 있어서 매체의 저장 공간 이용 효율성에 관한 평가를 수행하고자 한다. 평가에서 공간 효율성 비교는 두 기록 방식에 따라 광매체에 대한 기록을 한 결과, 사용자 데이타를 제외한 오버헤드 공간이 얼마나 소요되었는지를 비교하는 것을 의미한다.

공간 비교에서 본 논문의 방식과, 멀티 세션 방식에서 모두, 저장되는 단위 데이타의 크기는 항상 트랙의 최소 크기인 300섹터(600KB)를 넘는 것으로 가정한다. 또 중간 정보를 저장하기 위한 ISO-9660 메타 데이타의 크기는 파일 수와 디렉토리를 구성하는 방법에 따라 다르지만 최대 99개의 실시간 데이타를 저장할 때도 10KB를 넘지는 않으므로 10KB로 가정한다. 멀티 세션 방식에서는 pvd 영역이 16번 섹터에서 시작하므로 이 값을 40KB로 가정한다. 표준에서 광매체 기록에서 각 세션은 각각 9MB와 13.5MB의 리드-인, 리드-아웃 영역을 가지며, 각 트랙 사이에는 1.2MB의 공간이 필요하다[2]. 위와 같은 값을 이용하여 계산할 때, 실시간 기록 대상 데이타의 개수가 N일 때, 멀티세션 방식의 경우 매체 공간 오버헤드는 식 (1)에 의해 계산될 수 있다.

$$Overhead_{multi\text{-}session} = N * (9MB + 13.5MB + 1.2MB + 40KB) - 9MB \quad (1)$$

식의 제일 뒤쪽에서 9MB를 뺀 것은 첫 번째 리드-인 영역이 논리 섹터 0번 이하의 위치에 저장되기 때문이다. 또 이 논문이 제안하는 실시간 기록 방식의 경우 매체 공간 오버헤드는 식 (2)에 의해 계산된다.

$$Overhead_{proposed} = R + N * (1.2MB + 10KB) + 13.5MB \quad (2)$$

식에서 R은 본 논문의 방식에서 최종 ISO-9660 메타 데이타를 위해 예약된 첫 트랙의 크기로 300섹터(600KB)에 해당된다. 본 논문의 방식에 따라 만들어진 매체는 하나의 세션으로 구성되기 때문에, 세션 오버헤드는 리드-아웃 영역 한 번만 계산된다.

그림 6은 몇 개의 실시간 데이타 개수에 따른 두 방식의 공간 오버헤드 크기를 비교한 결과이다. 그림에서 본 논문의 방식은 부가적인 예약 트랙의 오버헤드가 존재함에도 불구하고 단일 세션 구조로 이루어졌다는 장점 때문에 공간 오버헤드가 적어 저장 공간 이용에 있어 매우 효율적임을 알 수 있다. 특히 멀티 세션 방식의 경우, 저장할 실시간 데이타의 개수가 30인 경우, 오버헤드의 크기만으로도 통상적인 cd의 저장 한계인 700MB를 초과하고 있으며, 15개의 데이타를 기록할 때 cd 용량의 반 정도만 이용 가능하다. 또 dvd를 사용할 때 역시 98개의 데이타를 기록하는 경우, 용량의 반 정도를 오버헤드로 소비한다. 반면에 본 논문에서 제안하는 실시간 기록 방식의 경우, 데이타 개수가 한계치인 98개 일 때에도 저장 공간 오버헤드가 cd의 경우 19.0%, dvd의 경우 2.8%에 불과하다.

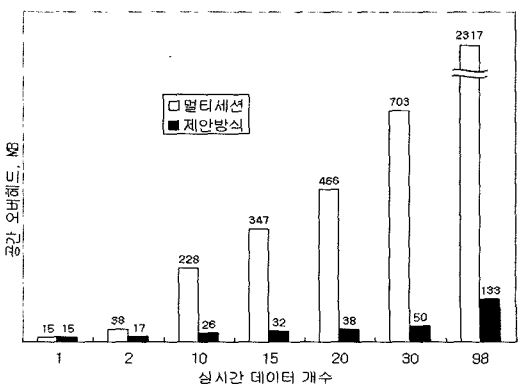

그림 6 공간 효율성 비교

4.5 응용 프로그램의 구현과 임베디드 시스템에의 적용

본 연구에서는, 구현된 API를 시험하고 공개 소스로 개발된 API의 활용도를 높이기 위하여, 이미 잘 알려진 광매체 기록 프로그램인 *cdrecord*의 명령어 인터페이스를 그대로 적용한 텍스트 모드 기록 프로그램을 개발하여, 기존의 광매체 기록을 위한 GUI 전처리기 프로그램들을 수정 없이 사용할 수 있도록 하였다. 또 구현된 API의 효용성을 검증하고, 응용 프로그램에서 개발된 API를 직접 이용하는 예를 보이기 위하여 공개 소스 프로그램인 *GnomeBaker*를 기반으로 그래픽 기록 프로그램을 제작하였다. 두 응용 프로그램은 모두 리눅스 상에서 구현되었다. 또 실시간 기록 시나리오를 따르는 응용 프로그램은 리눅스에서 프로토타입 버전이 만들어진 뒤, 다른 계층의 광매체 API들와 함께 대표적인 실시간 운영체제인 pSOS에 이식되어, 이를 운영체제로 사용하는 상용 dvr에 적용되었다.

4.6 개발된 API의 문제점과 향후 과제

이 논문에서 개발된 광매체 접근 API와 그를 이용한 실시간 기록 API는 모든 응용에 적합한 완벽한 것이 아니다. 개발된 API는 다음과 같은 문제점을 가지고 있으며, 추가적인 개발이나 광매체에 관한 표준과의 호환성을 유지하는 우회적인 방법에 대한 연구로 해결될 수 있을 것이다.

- 실시간 기록의 중간 상태에 있는 매체에 대한 인식 불능 : 이 문제는 이 논문에서 개발된 API뿐만 아니라 모든 트랙 단위 기록 프로그램의 문제이기도 하다. 하나 이상의 트랙(실시간 데이타)이 기록된 상태에서, 쓰기 완료 명령을 거치지 않은 상태의 광매체는 다른 일반적인 컴퓨터 시스템에서 정상적인 매체로 인식되지 않는다. 물론 개발된 API를 이용하는 프로그램, 임베디드 시스템이나 대부분의 상용 광매체 기록 프로그램들은 이 중간 상태의 매체도 읽을 수 있다. 하지만, 리드-인, 리드-아웃 영역이 결여된 이 상태의 매체를, 기록 완료된 매체만을 인식하는 범용 운영체제에서는 인식하지 못한다. 이 문제는 표준을 유지하면서 효율적인 실시간 기록을 하는 경우, 수용할 수밖에 없는 것으로 실시간 기록을 위한 레이아웃 표준이 새로 제정되거나, 이 레이아웃을 가진 미완성 매체를 읽기 위한 소프트웨어를 각 운영체제마다 제공함으로써 해결 가능하다.
- 다양한 매체 레이아웃 지원 : 광매체 표준에서는 멀티미디어 데이타 지원을 위한 다양한 매체 레이아웃을 지원하고 있다. 특히 svcd나 dvd 비디오 형식은 널리 보급된 일반 dvd 플레이어를 이용하여 바로 재생이 가능하기 때문에, 실시간 영상 기록의 좋은 레이아웃 방식이 될 수 있다. 하지만 이들 표준 레이아웃은 메타 데이타가 사전에 고정되어야 하는 문제 때문에 실시간 기록에 대한 직접적인 지원이 어려운 경우가 많다. 따라서, 이 논문이 ISO-9660 파일 시스템에 대하여 접근하는 것과 유사한, 여러 우회적인 방법으로 효율적인 실시간 지원 방법을 찾는 연구가 필요할 것으로 본다.
- 패킷 모드 기록에 대한 지원 : 패킷 모드 기록 방식은 트랙 기록 방식에 비하여 매우 느리며, 대부분의 운영체제가 기본으로 지원하지 않고 있다. 하지만, 플로피 디스크와 같은 편리함 때문에, 캠코더와 같은 소비자 지향적인 임베디드 시스템에 유망한 매체라고 볼 수 있다. 따라서 cdrw, dvd±rw, dvd-ram 매체를 위하여 기 개발된 API에 패킷 모드 기록 방식 지원을 위한 API를 추가하고, 레이아웃 API 계층에도 fat, udf와 같은 파일 시스템 레이아웃에 대한 지원이 필요하다고 사료된다.

5. 결 론

본 논문에서는 계층적 설계에 기반을 둔 새로운 공개 소스 광매체 API의 개발 결과를 기술하였다. 개발된 광매체 API는 기존의 많은 광매체 기록 프로그램들이 이용하는 방식의 데이타 백업 응용에 그대로 사용될 수 있다. 또 논문에서는 효율적인 실시간 데이타 기록을 위한 새로운 광매체 레이아웃을 제안하고, 실시간 기록을 위한 상위 API와 기록 시나리오도 같이 개발하였다. 제안된 레이아웃은 표준인 ISO-9660 파일 시스템 레이아웃을 따르기 때문에 모든 운영체제에서 인식이 가능한 호환성을 가진다. 또 논문에서는 실시간 저장에 따른 저장 공간 오버헤드 비교를 통하여, 제안된 실시간 기록 방식의 공간 이용 효율성이 멀티 세션 기록 방식에 비하여 월등이 우수함을 보였다.

연구에서는 커널 버전 2.6이 실행되는 리눅스 환경에 개발된 API를 적용하고, 기존 광매체 프로그램을 대치할 수 있는 텍스트 모드 응용 프로그램과, 개발된 API를 직접 이용하는 그래픽 응용 프로그램을 구현함으로써 그 유용성과 호환성을 확인하였다. 또 개발된 API는 실시간 운영체제인 pSOS를 사용하는 상용 임베디드 시스템인 dvr 등에서 광매체에 실시간으로 영상을 기록하는 용도로 적용되어 그 가치를 입증하였다.

참 고 문 헌

[1] *Compact Disk Digital Audio System*, IEC-908 (Red Book), 1987, ISO

[2] *Data Interchange on Read-only 120 mm Optical Data Disks*, ISO/IEC-10149 (Yellow Book), 1995, ISO.

[3] DVD Forum 홈페이지, http://www.dvdforum.org
[4] *Volume and File Structure of cdrom for Information Interchange*, ISO/IEC-9660, 1999, ISO.
[5] *Universal Disk Format Specification (UDF)*, Rev. 2.6, Optical Storage Technology Association, March 2005.
[6] *Mt. Fuji Commands for Multimedia Devices*, American National Standard Institute NCITS T10/99-121R0, 1999, ANSI.
[7] *Small Computer System Interface*, American National Standard Institute X3T9.2/375R, ANSI.
[8] *AT Attachment with Packet Interface 8 (ATA/ATAPI-8)*, American National Standard Institute NCITS T13/1532D, ANSI.
[9] *Advanced SCSI Programming Interface*, Adaptec, 2001.
[10] CDRtools 홈페이지, http://cdrecord.berlios.de/old/private/cdrecord.html
[11] Software Architects Inc. 홈페이지, http://www.udftoolkit.com/
[12] mkisofs 홈페이지, http://www.andante.org/mkisofs.html
[13] Nero Inc. 홈페이지, http://www.nero.com
[14] GnomeBaker 홈페이지, http://gnomebaker.sourceforge.org

이 민 석

1986년 서울대학교 컴퓨터공학과(학사) 1988년 서울대학교 컴퓨터공학과(석사) 1995년 서울대학교 컴퓨터공학과(박사) 1999년~2002년 (주) 팜팜테크 CTO. 1995년~현재 한성대학교 컴퓨터공학과 교수 관심분야는 임베디드 시스템, 파일 시스템, 공개 소스

송 진 석

2005년 한성대학교 컴퓨터공학과 졸업(학사). 2007년 한성대학교 컴퓨터공학과 졸업(석사). 관심분야는 임베디드 시스템, 파일 시스템, 스케줄링 알고리즘

윤 찬 희

2005년 한성대학교 멀티미디어학과(학사) 2007년 한성대학교 컴퓨터공학과(석사) 관심분야는 임베디드 시스템, 임베디드 리눅스, 스케줄링 알고리즘, 네트워크 QoS