임베디드 시스템을 위한 UDF 파일 시스템 구현

임용관, 이호송°, 성영락†, 이철훈, 권택근
충남대학교 컴퓨터공학과, † 국민대학교 전자정보통신공학부

Implementation of UDF File System for Embedded System

Yong-Gwan Lim, Ho-Song Lee°, Yeong Rak Seong†, Cheol-Hoon Lee, and Taeck-Geun Kwon
Dept. of Computer Engineering, Chungnam National Univ.
† School of Electrical Engineering, Kookmin Univ.
{yklim, hslee, chlee, tgkwon}@ce.cnu.ac.kr, † yeong@mail.kookmin.ac.kr

요 약

멀티미디어에 대한 사회적 욕구가 한층 커져감에 따라, 보다 고용량 고효율의 매체에 대한 연구도 활기차게 진행되어 왔다. 그리고 그 미디어에 적합한 파일시스템을 개발 하는 일 또한 꾸준히 진행되어 온 일이다. 여러 파일 시스템 중 하나인 UDF 는 DVD 매체를 위한 파일 시스템으로 사용되고 있다. 본 논문에서는 DVD 플레이어에 탑재하기 위해 삼성전자에서 개발한 ARM920T 마이크로 프로세서 코어를 사용하는 임베디드 시스템에 탑재될 실시간 운영체제를 위한 UDF 파일 시스템을 구현하였다. 이 UDF 파일 시스템은 가상 파일 시스템과 연동한다.

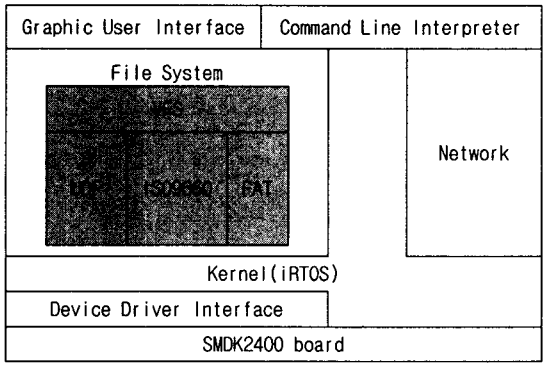
1. 서 론

컴퓨터 하드웨어 기술의 발달과 어울러 각종 인터넷과 정보통신 서비스의 발달은 각종 사무환경, 교육, 정보 검색 등의 분야에서 개인용 컴퓨터의 보급을 촉진시켰다. 이러한 개인용 컴퓨터의 확대 보급은 보다 고용량, 고효율의 멀티미디어 서비스의 수요가 확대됨을 의미하고, 이러한 서비스를 적절하게 사용자에게 제공해주기 위해 각종 저장 매체들의 개발이 필요하게 되었다. DVD 는 21 세기 차세대 영상 매체로서 각광 받고 있고 그 보급률과 시장은 해가 거듭할 수록 증가하고 있는데, 이러한 DVD 매체에서 사용하고 있는 파일시스템이 ECMA167(ISO13346)을 규격화한 UDF(Universal Disk Format)이다. 한편 이러한 UDF 포맷으로 저장 된 매체를 임베디드 시스템에서 접근하려 한다면, 임베디드 시스템의 목적 및 특성에 부합하는 기능을 파일시스템 설계시 고려해 주어야 한다. 임베디드 시스템은 일반적인 시스템에 비해 여러 자원이 부족하고, 그 성능 또한 일반 시스템에 비해 상대적으로 좋지 않게 된다. 그러므로 이러한 임베디드 시스템에서의 파일 시스템 구축은 일반적인 시스템에서의 그것 보다도 짜임새 있고 내부 자원을 효율적으로 사용할 수 있도록 설계해 주어야 한다. 본 논문에서는 삼성전자에서 개발한 ARM920T 마이크로 프로세서 코어를 사용하는 임베디드 시스템에 적합한 UDF 파일 시스템을 구현하였다. 본 논문의 구성은 다음과 같다. 2 장에서는 본 논문을 구성하고 있는 전체 시스템 환경 및 가상 파일 시스템에 대해 간략하게 소개한다. 3 장에서는 UDF 의 표준 및 표준에 나와있는 여러 자료구조에 대해 소개하고. 4 장에서는 구현한 UDF 에 대해서 기술한 후 마지막으로 5 장에서는 결론 및 향후 연구 과제에 대해 기술한다.

2. 전체 시스템 환경

2.1 Overview

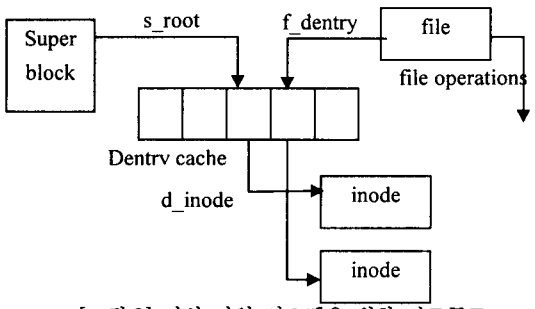
위 [그림 1]에서 보는 바와 같이 최하위에 개발 환경이 되는 보드 위에서 실시간 운영체제의 커널 부분이 동작하게 되고, 각각의 파일시스템은 이 커널 및 가상파일시스템과 연동하여 파일 접근에 필요한 오퍼레이션들을 제공해 준다.



[그림 1] 전체 시스템 구성도

2.2 가상 파일 시스템과 자료구조

일반적으로 파일 시스템은 여러 종류의 저장 장치들을 지원하게 된다. 이러한 여러 저장 장치에 사용자가 파일을 저장할 경우, 각 저장 장치에서 사용하는 파일 시스템에 상관없이 통일된 인터페이스를 통해 작업을 하도록 하고 실제 동작은 파일 시스템이 저장 장치에 적합한 함수를 호출하도록 해주는 것이 가상 파일 시스템이다.



[그림 2] 가상 파일 시스템을 위한 자료구조

또한 버퍼 캐쉬, 수퍼블럭, 디렉토리엔트리, 아이노드와 같이 공용으로 사용해야 하는 자료구조에 대한 설계도 동일해야 한다. [그림 2]는 본 논문에서 구현된 자료구조도이다. 이 구조도는 어떤 파일을 오픈 하였을 때 file 구조체가 생성되고 그와 연관된 디렉토리 엔트리와, 아이노드가 어떤 연관관계를 가지는지에 대해 보여주고 있다. Superblock 의 s_root 가 가리키는 것은 디렉토리 캐쉬에 있는 루트 디렉토리이다. 이 루트디렉토리는 그 디렉토리와 연관된 아이노드, 즉 루트디렉토리의 아이노드를 갖게 된다. 파일을 오픈하면 file 구조체가 생성되어 오픈된 파일과 연관된 디렉토리 엔트리와 아이노드를 찾아갈 수 있게 된다.

2.3 가상 파일 시스템과 UDF 의 파일 오퍼레이션 연동

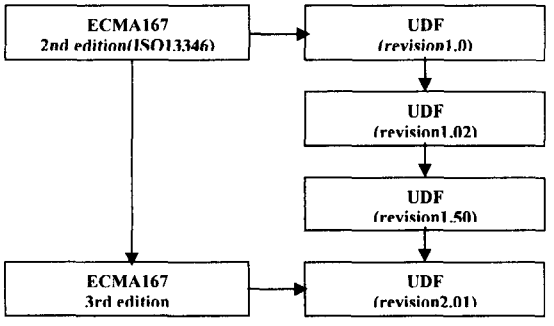
가상 파일 시스템을 지원하기 위해서는 각각의 파일 시스템이 표준화된 파일 오퍼레이션 API 를 따라야 한다. 사용자의 편의를 위해 [표 1]에서 보는 바와 같이 VFS 에서 표준 API 를 제공하고, 그에 매치되는 UDF 의 파일 오퍼레이션을 제공하여, 사용자가 선택한 적절한 파일 시스템의 파일 오퍼레이션이 수행 되도록 설계되었다. 위 표에 언급되지 않은 llseek 과 같은 나머지 파일 오퍼레이션들은 각각의 파일 시스템이 따로 구현해주지 않아도 가상 파일 시스템 레벨에서 공통적으로 제공해 줄 수 있도록 설계하였다. 만약 가상 파일시스템을 사용하지 않고, UDF 만 단독으로 사용되는 시스템을 설계하려면 이러한 오퍼레이션들도 모두 구현해 주어야 한다.

[표 1] 가상파일시스템과 UDF 의 파일 오퍼레이션

VFS(표준 API)	UDF
Read	Udf_file_read
Readdir	Udf_readdir
Open	Udf_open_file

3. UDF

3.1 UDF 표준



[그림 3] UDF 관련 표준

ECMA167 은 비순차적인 기록을 사용하는 한번쓰기 그리고 재기록가능한 매체를 위한 볼륨과 파일 스트럭쳐에 대한 유럽컴퓨터제작자 협회의 표준이다. [그림 3]에서 보는 바와 같이 이 표준은 *ISO13346* 으로 국제 표준화 되었으며, 추후에 이 표준들은 OSTA(Optical Storage Technology Association)에 의해 자료 교환을 증대시키고, 복잡성을 최소화해 다시 UDF 란 이름으로 표준화 되어 DVD 매체에서 사용될 파일 시스템이 되었다.

DVD-ROM 은 CD-ROM 매체와의 호환성을 고려하여 UDF 브릿지 볼륨/파일구조를 채용한 UDF(1.02)를 기반으로 작성되었고, DVD-R 과 DVD-RAM 은 CD-UDF 구조를 채용한 UDF(1.50)버전을 채용하였다. 본 시스템에서는 가장 최근에 발표된 UDF2.01 을 채용하였는데, UDF 는 하향호환성을 지원하므로 본 시스템에서 구현한 UDF 는 현존하는 모든 DVD 매체를 지원한다.

3.2 볼륨 및 파일 스트럭쳐 및 디스크립터

UDF 는 기록된 매체의 유용한 정보들을 디스크립터란 이름으로 형식화 해 놓고 있다. 본 논문에서는 그 디스크립터들 중 파일 읽기에 꼭 필요한 몇몇 디스크립터 및 볼륨 스트럭쳐들을 살펴본다.

3.2.1 Volume Recognition Sequence(Area)

논리적 섹터 16 번째 위치에 기록된 정보로써 매체를 구분해 주기위한 정보가 기록되어 있다. 예를 들어 DVD 매체일 경우엔 stdIdent[0]필드에 " BEA01" 이란 정보가 기록되어 있고, CD-ROM 미디어일 경우 " CD0001" 이란 정보가 기록되어 있다.

3.2.2 Anchor Volume Descriptor Pointer

순차적으로 기록된 디스크립터들의 모임을 Volume Descriptor Sequence 라 한다. Anchor Volume Descriptor Pointer 는 매체의 256 번째 섹터, 또는 마지막섹터, 또는 마지막섹터에서 256 번째 이전 섹터에 기록되어 Volume Descriptor Sequence 의 위치를 가지고 있다. 각각의 Descriptor 들에게는 Descriptor Tag 가 있어 각각의 Descriptor 들을 구분해 준다.

3.2.3 Primary Volume Descriptor

Volume Identifier 나 Volume Set Identifier 와 같이 매체에 기록된 Volume 에 대한 기본적인 정보들을 기록해 놓은 디스크립터이다.

3.2.4 Logical Volume Descriptor

논리적 볼륨의 논리 블록 사이즈, Partition Volume Descriptor 에 있는 Partition_location 필드와 이 Logical Volume Descriptor 에 있는 FSD_Location 필드를 더한 값의 위치에 FileSet Descriptor 가 있다.

3.2.5 Partition Volume Descriptor

볼륨의 파티션을 구분해주고, 파티션에 필요한 애트리뷰트들을 정의해 놓고 있다.

3.2.6 FileSet Descriptor

볼륨의 파일셋 식별자를 가지고 있고, 루트 디렉토리의 FileEntry 의 위치를 가지고 있는 rootDirectoryICB 필드 등을 가지고 있다.

3.2.7 FileEntry Descriptor

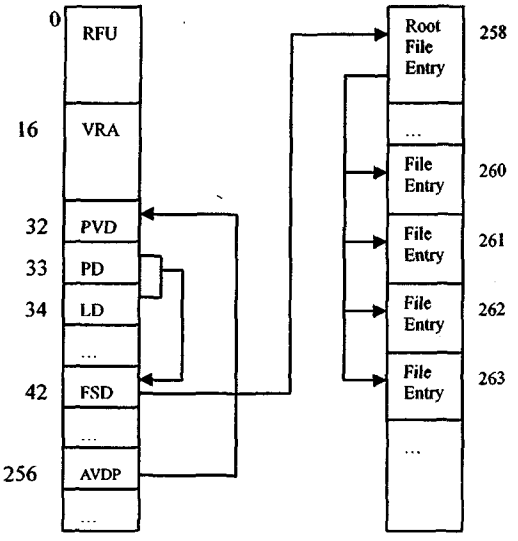
하나의 파일 또는 디렉토리당 하나의 FileEntry Desciptor 가 있으며, 각 파일/디렉토리에 해당하는 아이노드에 대한 정보들이 저장되어 있다. 또한 이 파일에 할당된 레코드의 크기를 가지고 있고, StrategyType 필드에 의해 레코드의 할당 방법 또한 알 수 있다.

[그림 4]는 위에서 설명한 디스크립터들을 조합하여 루트 디렉토리의 파일엔트리를 찾는 과정과 루트디렉토리의 자식 화일들에 접근하는 예에 대한 구조도이다

4. 구현 내용

4.1 단독 UDF 파일 시스템

가상 파일 시스템과 실시간 운영체제가 연동하는 시스템환경에서 UDF 파일 시스템을 구현하기에 앞서, 단독으로 실행가능한 UDF 파일 시스템과 쉘을 리눅스 환경에서 먼저 구현하였다.



[그림 4] UDF 의 물리적 자료구조

UDF 매체의 이미지를 읽어들여 파일로 저장하고 그 파일을 표준화일입력으로 받아들여 구현한 파일 오퍼레이션을 적용하였다. 각각의 파일 오퍼레이션들을 수행할 때 적절한 아이노드와 디렉토리 엔트리 오퍼레이션돌을 수행 할 수 있게 해주어 UDF 에 들어있는 자료구조돌이 메모리상에서 관리되게 하였다. DVD 플레이어에 탑재될 UDF 파일 시스템을 만드는 것이 목적이었기 때문에 update 나 delete 와 관련된 파일 오퍼레이션들은 구현하지 않았다.

쉘의 명령어는 ls, cat, chdir 과 같이 기본적인 명령어만 구현해 주었고, 오퍼레이션들은 [표 2]와 같다

[표 2] UDF 파일 시스템의 오퍼레이션

udf_read_super(struct super_block* sb, void* options)
시스템 초기화때에 매체에 대한 디스크립터 정보를 읽어들여 super_block 구조체의 필드를 채운다.
Udf_file_open(struct inode* inode, struct file* file)
Inode 에 해당하는 파일을 오픈하고자 할 때 file 구조체를 생성하고 해당하는 파일 디스크립터를 반환한다
udf_file_read(struct file* file)
File 에 해당하는 파일을 읽어들여 2048byte 크기의 캐쉬버퍼에 저장한다. 또는 캐쉬버퍼의 파일 내용을 읽어온다
udf_file_readpage(struct file* file, struct page* page)
File 에 해당하는 파일의 페이지를 읽어들여 캐쉬버퍼에 저장한후 page 가 그 캐쉬버퍼를 가리키게 한다.
Udf_file_Readdir(struct file* file,void *dirent, filldir_t filldir)
File 에 해당하는 디렉토리를 읽어 자식 파일과 디렉토리를 디렉토리 엔트리에 추가한다.
Udf_lookup(struct inode* inode, struct dentry* dentry)
Dentry 의 d_name 과 이름이 같은 아이노드를 inode 하위 디렉토리에서 검색한후 그 파일이나 디렉토리가 존재하면 dentry 를 채워서 리턴한다.
Udf_llseek(struct file* file, loff_t offset, int origin)
File->f_pos 을 origin 의 값에 따라 offset 만큼 이동해 준다.

4.2 임베디드 시스템을 위한 UDF 파일 시스템

앞서 설명한 바와 같이 가상 파일 시스템과 UDF 파일 시스템, 그리고 실시간 운영체제가 연동하는 환경에서 UDF 파일 시스템을 구현하였다. 구현한 오퍼레이션들은 [표 2]와 동일하지만 환경이 임베디드 시스템으로 바뀌었기 때문에 몇가지 고려사항이 추가되었다.

첫째로 파일을 접근하고자 할때의 메모리 관리 방법이다. 운영체제가 매체를 인식하였을 때(mount 시) 파일과 디렉토리 구조에 대한 아이노드와 디렉토리엔트리를 모두 메모리상으로 올려 놓는 방법을 사용하지 않고, 실제로 어떤 파일에 접근하고자 할때에만 그 파일과 그 파일이 속한 디렉토리 구조만을 메모리 상에 올려 놓아 보다 메모리를 적게 사용할 수 있도록 배려하였다.

둘째로 수퍼블럭, 아이노드, 디렉토리엔트리 등과 같은 자료구조의 최적화하였다. 임베디드 시스템에서는 멀티유저 등과 같이 기존의 시스템에서 지원해 주어야 하는 몇몇 요소들이 거의 쓸모가 없고, 오히려 이러한 요소를 지원해 주는 일은 메모리를 낭비하는 일이다. 이 시스템에서는 쓸모없는 자료구조들 간의 링크나 멀티유저와 관련된 모든 자료구조의 필드들을 제거하여 최적화하였다.

5. 결론 및 향후 연구과제

본 논문에서는 임베디드시스템을 위한 UDF 파일 시스템을 구현하였다. 이 파일 시스템은 가상 파일 시스템과 실시간 운영체제와 연동할 수 있게 설계되었다.

향후 연구 과제로는, 매체의 읽기뿐만 아니라 쓰기까지도 가능한 파일 오퍼레이션을 구현해 주는 것이다.

참고문헌

[1] UDF revision2.01, "*Universal Disk Format Specification*" ,Mar. 2000.

[2] ECMA-167, "*Volume and File Structure for Write-once and Rewritable Media using Non-Sequential Recording for Information Interchange 3rd edition*", June 1997.

[3] ISO-13346, "*Volume and File Structure of Write-once and Rewritable Media using Non-Sequential Recording for Information Interchange*", Dec. 1995.

[4] M Beck and H Bohme and M Dziadzka and U Kunitz and R Magnus and D Verworner, *Linux Kernel Internals 2nd edition* , Addison Wesley, Longman, 1998.

[5] Daniel P.Bovet and Marco Cesati, *Understanding the LINUX KERNEL 2nd edition*, O' Reilly, 2003.

[6] 조유근, 최종무, 홍지만, *커널 프로그래밍*, 교학사, 2002.

[7] 강석민, 송재영, 조정철, 권택근 "임베디드 시스템을 위한 파일 시스템 구현" 한국정보과학회 춘계학술발표 논문집 제 29 권 1 호, pp.61~63, 2002.