리눅스 파일시스템 템플릿 구성에 관한 연구

경주현*, 장명우+, 이민석*
*한성대학교 컴퓨터공학과, +(주)엠트론
e-mail: leaders3@naver.com, jmw@mtron.net, minsuk@hansung.ac.kr

A Study on the Linux Filesystem Template

Juhyun Kyung*, Myoungwoo Jang+, Minsuk Lee*
*Dept. of Computer Engineering, Hansung University, +Mtron Co. Ltd.

요 약

본 논문에서는 다양한 응용에 적용하기 위하여, 또는 연구를 위하여 새로운 파일시스템 개발을 개발하는데 있어서 초기에 겪어야 하는 기술적 어려움을 쉽게 극복하기 위한 파일시스템 템플릿을 개발하였다. 연구에서는 리눅스 파일시스템의 분석을 통해 파일시스템 구현에 필요한 세부 정보를 문서화하였고 쉽게 이용할 수 있는 파일시스템 템플릿을 구현하였다. 작성된 문서와 파일시스템 템플릿의 소스를 이용하여 새로운 파일시스템 개발을 필요로 하는 연구자들의 개발 생산성을 높일 수 있게 하고 부수적인 노력을 획기적으로 줄일 수 있다.

1. 서론

현대에는 다양한 멀티미디어 디바이스들의 수요와 생산이 나날이 증대됨에 따라 파일시스템 역시 다양한 목적으로 개발이 이루어지고 있다. 하지만, 그 개발 과정은 복잡하고 어려우며 모든 파일시스템에 공통적인 기본 요소에 대한 중복된 개발 노력이 반복되고 있다.

한편, 리눅스에서는 다양한 파일시스템을 지원하기 위해 파일시스템들의 추상 계층인 Virtual Filesystem(이하 VFS)를 제공한다. VFS 때문에 파일시스템의 개발이 용이하게 되었지만, VFS를 이용한다고 해도, 아직도 많은 프로그램을 작성해야 하며, VFS 자체에 관한 문서의 부족 등 여러 가지 이유로 리눅스에서의 파일시스템 개발에는 여전히 높은 기술적 장벽이 있다.

실제 많은 파일시스템들은 기본적인 기능이 유사하며, 거기에 특정 파일시스템만의 기능이 추가되어 제작되는 것이 보통이다. 하지만 VFS 계층에 속하지 않는 부분 즉 공통적이면서도 기본적인 기능들을 구현하기 위해서 많은 노력이 필요하다. 만일 VFS와 완벽한 호환이 되고 파일시스템의 기본적인 기능이 구현되어 있는 템플릿 파일시스템이 존재한다면, 파일시스템 개발에 필요한 노력을 줄일 수 있으며, 좀 더 상위 단계에서 개발을 시작할 수 있을 것이다.

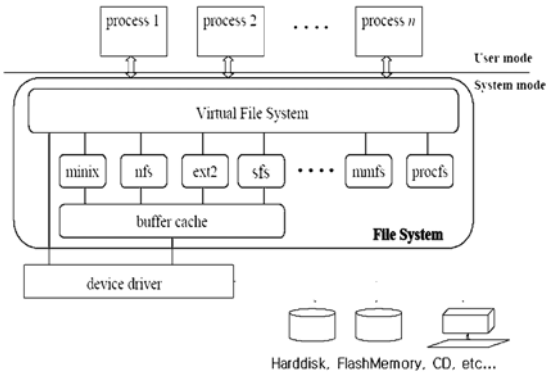
본 논문에서는 VFS와 특정 파일시스템과의 상호 작용을 면밀히 분석하여 간단한 디스크 레이아웃을 가지고 특정 파일시스템에 의존한 부분을 분리하여 이해가 쉬우면서도 완전히 동작하며 신뢰성 있는 파일시스템을 개발하여 각종 테스트를 통해 검증하고, 리눅스에서 다양한 파일시스템 개발을 용이하게 할 수 있는 리눅스 파일시스템 코드 템플릿과 그 개발 방법을 제안한다.

2. 연구목적

2.1 연구배경

2.1.1 리눅스 가상 파일시스템

VFS는 표준 유닉스 파일 시스템이 제공하는 모든 시스템 호출을 처리하는 커널 소프트웨어 계층이며, 다양한 파일 시스템에 대하여 공통의 인터페이스를 제공한다[1].



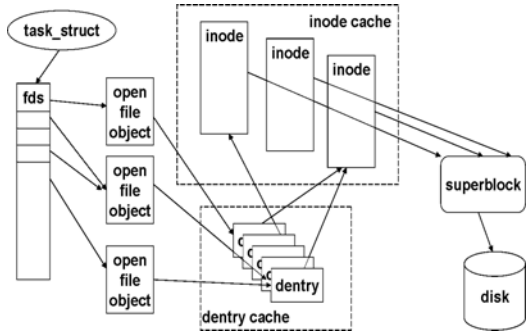
[그림 1] Virtual Filesystem Overview

[그림 1]에서, VFS는 기본적으로 응용 프로그램이 요청한 시스템 호출에 대한 인터페이스이며, 다양한 파일시스템들의 추상 계층이 되어, 어플리케이션은 파일시스템의 종류에 무관하게 공통의 시스템 호출을 이용하여 파일시스템을 사용할 수 있다. 예를 들어, 사용자는 DOS 파일시스템의 파일을 ext2 파일시스템으로 복사하려고 할 때 등, 파일시스템의 종류에 무관하게 동일한 시스템 호출을 이용할 수 있음을 의미한다. 리눅스는 VFS로 인해 다양한 파일시스템 지원이 가능하고, 최신의 커널 버전인 2.6에서

는 60여 가지의 파일시스템을 지원한다. 지원이 되는 파일시스템의 종류는 크게 3가지로 나눌 수 있다. 디스크 기반 파일시스템(예, Ex3, FAT32, ...), 네트워크 파일시스템(예, NFS, SMB, ...)과 가상 파일시스템이라고 불리기도 하는 특수 파일시스템이 있다. 본 논문에서는 디스크 기반의 파일시스템에 한해서 논의한다.

2.1.2 리눅스 가상 파일시스템의 구조

VFS는 공통 파일 모델을 지향하며 그것을 위해 4가지의 추상 데이터 타입을 사용한다. VFS는 이 4가지의 추상 객체로 구성되는 객체 지향 프레임웍이라고 볼 수 있다. 이 객체들은 superblock, inode, dentry, file 이며 [그림 2]는 이러한 객체들의 관계를 보여준다[2].



[그림 2] VFS 객체들 사이의 관계

2.2 기존연구

2.2.1 리눅스 가상파일시스템

현재 리눅스에서는 커널 수준에서 60여개의 파일시스템을 지원한다. 이렇게 다양한 지원이 가능한 것은 VFS가 추상화 계층으로서 잘 정의되어 있기 때문이다.

많은 리눅스 업체와 컴퓨터 업체에서도 리눅스용 파일시스템 혹은 자사의 파일시스템을 리눅스에 이식했다. IBM의 JFS와 CIFS, RedHat의 GFS2, Silicon Graphics의 XFS 등이 있고, 심지어는 마이크로소프트사의 FAT, NTFS와 같이 타 운영체제에서 만들어졌던 파일시스템들도 리눅스에 이식되었다.

또한 FUSE(filesystem for userspace)는 커널 모듈과 응응 프로그램 라이브러리를 제공하여 유저 레벨의 파일시스템의 제작을 가능하게 한다. 이것을 이용하면 간단한 코드만으로도 파일시스템을 만들 수 있다[8].

리눅스의 파일시스템은 다양한 목적과 방법으로 견고한 모습으로 발전하고 있으며 그 응용 또한 다양하다. 타 시스템에서 개발된 파일시스템을 VFS에 적용하여 동작하게 하는 부분도 잘 지원이 되어서 리눅스의 파일시스템의 종류는 갈수록 증가할 것이다.

2.2.2 국내의 파일시스템 연구

국내 리눅스 파일시스템 개발은 3가지 흐름이 있다. 첫째, 파일시스템 디자인 제안, 둘째 기존 파일시스템의 확장, 셋째 파일시스템의 구현이다[3][4][5][7].

디자인 제안이나 파일시스템의 확장에 관한 연구와는 달리 세 번째 파일시스템 개발 방향을 살펴보면 공통점을 가지고 있는데 그것은 리눅스의 VFS와는 독립적으로 파일시스템을 구현하고 있다는 것이다. 물론, 각 연구들은 각 연구 목적에 부합하는 파일시스템을 개발하였지만 하나의 운영체제 안에서 같은 역할을 하는 2개의 서브시스템을 가지고 있는 형태가 되었다. VFS에 무관하게 동작하는 파일시스템은 그 구조에 따라 다르지만 개발이 훨씬 단순할 수는 있다. 하지만 이런 장점이 있는 반면 여러 가지 단점도 있다. 먼저, 호환성이 부족하다. 표준 시스템 호출이 VFS에 요청하기 때문에 VFS에 무관하게 동작하는 파일시스템은 새로운 인터페이스를 응용 프로그램에 제공하여야 한다. 그리고 리눅스 운영체제는 역시 VFS를 통해 접근하지만, 이것이 파일시스템인지조차 인식할 수 없다. 또 자원을 효율적으로 사용할 수 없다. 파일시스템들이 공유할 수 있는 자원들을 2개의 서브시스템으로 나눠서 사용하여야 하므로 효율성이 떨어진다. 마지막으로 2개의 같은 시스템이 있으므로 이것에는 당연히 기능 중복의 문제가 있다. 같은 기능이 필요하더라도 재사용이 불가능하다.

2.3 리눅스 파일시스템 개발의 어려움

개념적으로는 VFS 계층은 특정 파일시스템과 분리가 될 수 있도록 설계되었으나, 구체적인 구현에 들어가면 계층의 분리가 잘 되어있지 않은 부분이 많아, 실제로는 인터페이스가 아닌 커널의 다른 부분과 VFS 내부의 동작을 많이 이해하여야만 이용이 가능하다. 그러므로 VFS의 소스코드를 면밀히 분석하지 않는 다면 파일시스템 개발 및 포팅은 매우 어려운 작업이 된다. 또한 관련 문서가 매우 부족하다.

2.4 연구의 필요성

파일시스템은 그 디스크 레이아웃 구성과 메타 데이터의 활용에 따라 그 모습을 달리한다. 하지만 VFS의 공통 인터페이스에 대한 부분은 호환성을 위해 꼭 구현해야 하는 부분이며 그 부분은 파일시스템에 따라 조금씩 차이는 있지만 같은 원리가 적용되므로 이해하기 쉬운 템플릿을 제공한다면 파일시스템의 개발 시작점으로서 공통의 코드의 중복을 피하여 개발의 오버헤드를 줄일 수 있다.

3. 연구내용

3.1 파일시스템 템플릿 디자인

이 연구에서 설계할 파일시스템은 특정 파일시스템의 개발 시작점이 될 수 있는 프레임웍을 제공한다. 이를 위해 리눅스에 운영되는 다양한 파일시스템들의 소스를 면밀하게 분석하였으며, 그 결과 템플릿으로 사용할 파일시

스템의 디자인 목표를 현존하는 파일시스템들의 특성 중에서 공통적인 부분들을 가능한 표현하고 표준 파일시스템 요구 사항(basic functionality)을 충실하게 구현하는 것으로 정하였다. 제안된 파일시스템은 다음의 세 가지를 기본 원칙으로 설계되었다.

- 가능한 단순한 구조를 유지한다.

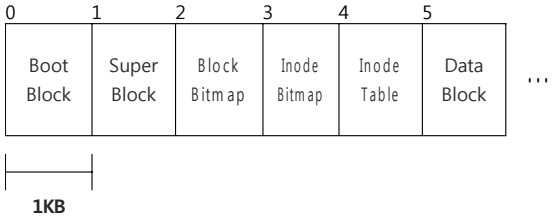
파일시스템이 가지고 있어야할 최소한의 메타데이터를 표현하는 범위 내에서 디스크 구성을 단순하게 하여 직관적 모습을 유지한다.

- UNIX의 파일 모델을 준수한다.

리눅스 VFS는 전통적인 Unix Common File Model을 그대로 따른다. 그러므로 디스크 구성 역시 가능한 동일하게 하여 설계와 구현의 외관적 차이를 줄여 개발을 용이하게 한다.

- 확장성을 고려한다.

본 파일시스템은 그 자체가 독립적인 파일시스템의 역할을 수행할 수 있고 본 시스템을 기본으로 하여 확장하는 것 또한 중요한 목표 이므로 디스크 구성에 유연성을 고려해 파일시스템의 복잡 도를 낮춘다.



[그림 3] 파일시스템 템플릿 디스크 레이아웃

[그림 3]은 파일 시스템 템플릿의 레이아웃을 보여준다. 부트 블록은 0본 블록에 위치하며 부팅 디스크가 아닐 경우라도 타 파일시스템들과의 호환성을 위하여 존재한다. 레이아웃에서 Bitmap과 Table들은 디스크의 용량에 따라 그 수가 가변적이므로 그림과는 달리 여러 블록들을 차지할 수 있다. 하지만 연구에서는 그림처럼 1개 블록을 차지하는 크기로 레이아웃을 설정하였다.

3.2 파일시스템 템플릿 구현

3.2.1 파일시스템 유틸리티 구현

구현된 파일시스템 유틸리티는 세 가지로 각각 파일시스템 초기화, 파일시스템 디버깅, 파일시스템 테스트의 역할을 수행한다.

파일시스템 초기화

파일시스템 초기화란 파일시스템의 레이아웃을 디스크에 쓰는 작업이며, 보통 *format*이라고 불리는 동작이다. 이것을 함으로써 디스크는 파일시스템으로서 사용될 준비가 되는 것이다. 초기화를 함으로써 디스크에 존재하던 기존의 정보는 지워지거나, 논리적인 방법으로는 접근할 수 없어, 더 이상 유효하지 않게 된다. 본 파일시스템에서는 *mkfs.sfs* 라는 이름의 초기화 프로그램을 구현했다. 파일시스템 초기화 프로그램은 레이아웃에 따라 다르게 작성하여야 하나 이 프로그램은 기존의 다른 파일시스템들의 초기화 프로그램을 축소하여 작성되었고 템플릿으로 사용 가능하다.

파일시스템 디버거

파일시스템의 초기화, 각종 파일 참조 동작이 성공적으로 이루어졌는지를 검증하기 위한 프로그램(*fsdb.sfs*)으로 superblock, inode 등 파일시스템 내의 메타 데이터의 정보를 확인하는 응용 프로그램이다.

파일시스템 테스터

또한 본 연구에서는 파일시스템의 동작을 테스트하는 응용 프로그램(*test.sfs*)을 구현하였다.

3.2.2 파일시스템 모듈의 구현

파일시스템 모듈은 커널 모듈로 파일시스템 템플릿 자체이다. 객체와 그 메쏘드로 구성되어 있고 파일시스템이 초기화 된 이후에 사용이 가능하다. 모듈이 본 연구의 핵심 코드로서 상세한 설명을 위해 각 소스 파일 별로 나누어 그 역할에 대해 설명한다.

소스 트리 구조는 다음과 같다.

sfs /include/s_fs.h
/sfs_module/bitmap.c
dir.c
file.c
inode.c
itree.c

- *s_fs.h* : 모듈과 유틸리티의 헤더 파일

파일시스템에서 사용할 매크로들과 데이터 구조, 함수의 정의가 작성되어있다. 여기서 파일시스템의 제한 사항을 기술하고 객체를 어떻게 구성할 것인지를 정의한다.

- *inode.c*

리눅스 커널의 동적 로드 가능 모듈로 구현된 파일 시스템 모듈을 커널에 등록하고 해제하는 과정과 수퍼 블록 오퍼레이션, 어드레스 스페이스 오퍼레이션이 구현되어 있다.

- *namei.c*

디렉터리 inode에 관한 inode 오퍼레이션이 구현되어 있다.

- *bitmap.c*

inode와 블록에 대한 bitmap처리가 구현 되어 있다. 사용 유무를 확인하고 사용할 bit에 대해 사용 중임을 표시하거나 해제될 bit에 대해 사용 해제를 표시한다.

- *dir.c*

디렉토리에 대한 파일 오퍼레이션이 구현되어 있다. *readdir()* 함수에 대한 구현이 되어 있다.

- *file.c*

file에 관한 파일 오퍼레이션과 inode 오퍼레이션이 구현되어 있다. 파일 오퍼레이션은 generic 펑션 즉 VFS에서 제공하는 함수들을 사용하기에 그 구현이 간단하다. 파일에 관한 I/O 역시 page단위로 이루어지기 때문에 address space 객체(*inode.c*)에서 처리한다.

- *itree.c*

inode의 블록 포인터에 대한 구현이 이루어진다. 본 템플릿에서는 직접 접근 블록만을 사용하기 때문에 간단하지만 큰 크기의 파일을 위해 간접 접근 블록을 사용하고자 한다면 이 파일에 그 부분을 추가 구현하여야 한다.

4. 파일시스템 테스트

4.1 파일 관련 쉘 명령과의 상호작용

리눅스 쉘에서는 여러 가지 파일 명령들을 제공한다. 파일 명령들은 실제 파일시스템들에 시스템 호출을 통한 요청을 한다. 유저 레벨에서 파일시스템의 동작을 가장 가시적으로 볼 수 있는 방법이다.

본 연구에서 테스트 대상이 되는 쉘 명령은 다음과 같다.

mount / umount : 파일시스템의 마운트 및 언마운트

ls : 디렉토리 내의 목록을 보여줌. 최초의 마운트 후 각 객체들이 잘 연결이 되어 수행되는 지를 확인

cp : 파일을 복사함으로써 읽기 및 쓰기 동작 상태를 확인

mkdir : 디렉터리 생성,

rmdir : 디렉터리 삭제

4.2 파일시스템 관련 시스템 호출 테스트

좀 더 세부적인 동작 유무를 확인하기 위하여 각종 시스템 호출하고 리턴값을 확인하는 방법으로 테스트한다. *read()/write()* 의 경우에는 실제 읽거나 써진 데이터를 확인한다. 결과는 [표 1]에 나타냈다.

System Call	Object	Support
mount()	FILESYSTEM	YES
umount()	FILESYSTEM	YES
mkdir()	DIRECTORY	YES
chdir()	DIRECTORY	YES
opendir()	DIRECTORY	YES
readdir()	DIRECTORY	YES
closedir()	DIRECTORY	YES
rmdir()	DIRECTORY	YES
creat()	FILE	YES
open()	FILE	YES
write()	FILE	YES
lseek()	FILE	YES
read()	FILE	YES
close()	FILE	YES

[표 1] 파일시스템 템플릿의 시스템 호출 지원 유무 및 시험 결과

5. 결론

본 연구에서는 문서와 파일시스템 템플릿 코드를 작성했으며 파일시스템의 개발 방법을 제공하여 리눅스 파일시스템의 기술 장벽을 낮추고자 하였다. 파일시스템에 관한 연구를 통해 개발에 필요한 모든 사항을 문서화하였고 파일시스템이 공통적으로 구현해야할 부분을 구분하여 작은 파일시스템 템플릿을 구현하였다. 본 연구의 결과는 인터넷 홈페이지를 통해 공개하였다. 파일시스템 연구자와 새로운 파일시스템을 개발하려는 개발자들은 이 연구의 결과 문서들과 파일시스템 템플릿을 가지고 파일시스템 개발에 필요한 사전 작업의 대다수를 피할 수 있으며, 파일시스템 자체의 전략에만 집중할 수 있게 되었다.

참고문헌

[1] Daniel Plerre Bovet, Marco Cesati, Understanding the Linux Kernel(3/E), O'REILLY, 2005

[2] Richard Grooch, Overview of the Linux Virtual File System, 1999

[3] 권우일, 윤미현, 이동준, 장재혁, 양승민, DVR 시스템을 위한 저널링 파일 시스템의 성능평가, 2002 추계 한국정보과학회 논문지, 2002.

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

[5] 원유집, 박진연, "정보가전용 멀티미디어 파일 시스템 기술" 멀티미디어와 정보화 사회, 한국과학기술진흥재단,

[6] 원유집 외, "멀티미디어 스트림의 QoS를 보장하는 통합형 파일시스템", 한국 정보 과학회, 2000

[7] 이민석, "대형 멀티미디어 파일을 위한 파일 시스템 구현", Journal of Information Technology Application & Management, vol.10, No. 4, pp. 169-183, 2003.

[8] 리눅스 커널 소스 공식 사이트 www.kernel.org