제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

Real-Time Linux 에서 소규모 파일 관리를 위한 파일 시스템의 설계

최용식*, 이상락**, 신승호***
인천대학교 컴퓨터공학과
e-mail: mars@incheon.ac.kr *

Design of File System for small File Management at Real-Time Linux

Yong-Sik Choi, Sang-rak, Seung-ho Shin
Dept. of Computer Engineering, University of incheon

요 약

리눅스 파일 시스템은 자기식 회전 디스크를 기반으로 개발되어 현재 임베디드 시스템에서 많이 사용되고 있는 플래시 메모리에 적용하기에 많은 문제점이 있다. 본 논문에서는 플래시 메모리의 특성을 고려하여 블록 사이즈를 재조정 함으로써 소규모 파일 시스템 관리에 유용하며 사용 빈도를 나타내는 블록을 추가하여 지움 정책, 파일 시스템 성능 향상을 위한 버퍼 캐쉬 기법에 의한 소규모 파일 시스템 모델을 설계하였다.

1. 서론

정보통신 분야의 빠른 발전에 힘입어 관련 분야에서 연구가 진행되고 있다. 그 중에서 Post-Pc 분야의 발전는 마이크로 프로세서 발전에 힘입어 다양한 형태의 임베디드 시스템의 발전을 가져왔다.

이러한 형태의 시스템으로는 홈오토메이션, 모바일 디바이스, 오디오, 자동항법, 디지털 이미지 시스템 등이 있다. 위와 같은 시스템은 많은 개발 비용이 요구되며 표준이 없고 서로 다른 하드웨어 환경에서 작동함으로써 주변 장치 드라이버에 대한 지원 여부가 불투명하다. 이에 저렴한 개발 비용을 가지면서 안정적인 운영체제의 필요성이 요구되고 있으며 최근 주목 받고 있는 운영체제가 Real-Time Linux 이다.

Real-Time Linux 는 커널 소스가 공개되어 있으며, 운영체제의 안정성과 신뢰성이 검증되어 있다. 또한 개발 도구가 모두 개방되어 있기 때문에 개발 비용이 적게 들어 경쟁력을 높일 수 있다는 장점이 있다.

그러나 임베디드 시스템의 특성상 여러 가지 제약요소를 가지고 있고, 기존의 리눅스 커널과 호환이 되지 않는 등 여러 가지 문제점을 가지고 있다. 그 중에서 가장 큰 차이점은 다음과 같다.

임베디드 시스템은 휴대성과 실시간성의 특징을 가지고 있기 때문에 부피와 무개가 크고 소비전력이 큰 장치 사용을 지양한다. 그렇기 때문에 저장 장치의 사용도 부피와 무게가 크고 소비 전력이 큰 자기식 회전 디스크를 사용하는데 비 효율적이다. 따라서 부피와 무게가 작고, 소비전력이 적으면서 빠르고 쉽게 저장할 수 있고 전원이 커진 상태에서도 기록 가능한 플래시 메모리(Flash Memory)를 사용하여야 한다. 하지만 플래시 메모리는 지움 정책 때문에 데이터 수정시 시간이 오래 걸리며, 수명이 영구적이지 못하다는 단점이 있다. 또한 임베디드 운영체제의 파일 시스템은 임베디드 운영체제의 특성과 플래시 메모리의 특성을 고려하여 설계되어야 한다.

본 연구에서는 운영체제의 기본적이며 중요한 역할을 수행하는 파일 시스템에 관하여 연구하고 Real-Time Linux 에서 효율적으로 동작할 수 있은 파일시스템을 설계하고자 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 리눅스 파일 시스템과 플래시 메모리의 특성 및 구조를 설명

* 본 연구는 한국과학재단 지정 인천대학교 동북아물류연구센터의 지원에 의한 것임.

하고, 3 장에서는 파일 시스템의 설계시 고려 사항 및 제반 사항에 관련된 내용을 제시하고 플래시 메모리를 이용한 소규모 파일 시스템의 구조, 메타데이터의 구조를 제안하려 한다. 마지막으로 4 장에서는 결론 및 향후 연구 계획에 대하여 논한다.

2. 관련연구

내장형 시스템의 저장매체로는 기존의 회전식 자기 디스크와 플래시 메모리가 사용된다. 플래시 메모리는 외부 충격에 강하고, 저전력으로 구동이 가능하며, 크기가 작은 장점을 가지고 있기 때문에 점차 그 사용이 증가하고 있다. 그러나 데이터의 내용을 수정하는데 시간이 오래 걸리며 수명이 영구적이지 못한 단점이 있다. 현재 리눅스 파일 시스템과 관련하여 사용되고 있는 것은 아래와 같다. 회전식 자기 디스크를 위한 파일 시스템은 ReiserFS, XFS, JFS, GFS, EXT3 등이 있으며 플래시 파일 시스템으로는 FTL(Flash Translation Layer), TrusFFS, JFFS, QplusFFS 등이 있다.

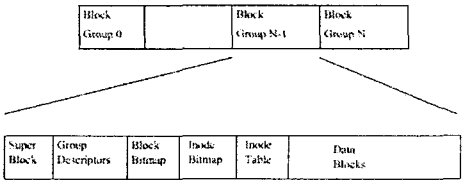
리눅스 파일 시스템은 소프트웨어에 의해 변환되어서, 모든 파일 시스템이 리눅스 커널의 다른 부분들과 그 위에서 실행되는 프로그램으로 보이게 해준다. 파일 시스템은 커널 내부에 포함되거나 모듈로 되어있으며, 커널은 디바이스 드라이버를 통해서 일정한 인터페이스를 제공 받아 제어 하게 된다.

2.1 리눅스 파일 시스템

초기의 리눅스 파일 시스템은 미닉스의 파일 시스템을 토대로 구성되었다. 하지만 이 파일 시스템은 최대 파일 크기 64Mbyte 로 제안되었고 파일 이름은 14byte 까지로 제안되고 속도가 느리다는 단점이 있었다. 이러한 점을 보안하여 1992 년 EXT(Extend File System)이 개발되었다.

이 파일 시스템은 최대 파일 크기는 2Gbyte 이고, 파일 이름은 255byte 까지 허용되었으나 미닉스 파일 시스템보다 더 느리며 단편화 현상이 발생하여 문제가 되었다. 이러한 문제점을 개선하여 1993 년 Remy Card 가 EXT2 을 개발하였으며 이 파일 시스템은 현재 리눅스의 표준 파일 시스템으로 사용되고 있다.

리눅스 파일 시스템은 가상파일시스템 (Virtual File System)을 공통의 인터페이스 계층으로 사용하여 커널이 단일화된 접근을 함으로써 xia, msdos, usmdos, iso9660, nfs, hpfs, sysv 등의 다양한 파일 시스템을 지원할 수 있다.



(그림 1) EXT2 파일 시스템의 물리적 구성도

EXT2 파일 시스템은 파일에 들어있는 데이터는 데이터 블록에 저장된다는 것을 전제로 하며 4KB 당 1 개의 Inode 를 할당한다.

이 파일 시스템은 파티션을 여러 개의 블록 그룹으로 나누고 각 블록 그룹은 수퍼블록의 복사본과 inode, 그리고 테이블 블록들을 가진다. 각 블록 그룹은 데이터 블록들을 그들의 inode 에 가깝게 유지시키고, 파일의 indoe 를 그들의 디렉토리에 가깝게 위치시키도록 하는 정보를 유지하여, 파일을 위치시키는 시간을 최소로 줄이고 데이터 접근 속도를 빠르게 해준다

2.2 플래시 파일 시스템

플래시 메모리는 셀(Cell)을 구성하는 구조에 따라 NOR 플래시, NAND 플래시, AND 플래시로 나눌 수 있으며 NOR, NAND 플래시를 주로 사용한다.

플래시 메모리는 하드디스크에 비해 견고하고, 비휘발성 특징을 가지며, 접근 시간이 빠르며, 크기가 작으며 저전력으로 작동한다. 그러나 플래시 메모리는 데이터를 한 번 쓴 영역에는 지움 과정을 수행하기 전까지는 다시 쓸 수 없다. 지우는 속도가 읽는 속도에 비해 느리고 지우는 횟수가 제어되어 있으며 지울 때에는 지움 단위(Erase Unit, EU) 또는 세그먼트(Segment) 단위로 지운다.

플래시 메모리에서 가장 큰 문제는 지우는 횟수의 제한과 지우는 속도가 상대적으로 읽는 속도보다 느린 데 있다. 따라서 지움 정책 (Cleaning Policy)이 중요한 문제이다. 지움 정책을 구성하는 가장 쉬운 방법은 로그 파일 시스템(Log-Structured File System, LFS)을 사용하는 것이다. LFS 에서 모든 저장은 순차적으로 일어난다. 저장매체를 끝까지 다 쓴 경우에는 처음으로 돌아와서 빈 공간을 확보해야 한다. 이때 새로운 데이터를 저장할 공간에 유용한 데이터가 있으면 다른 공간에 옮기고 데이터를 저장한다. 플래시 메모리의 경우 EU (Erase Unit) 단위로 지울 수 있다. 이러한 과정을 지움 정책이라 한다.

또다른 방법으로 Greedy 방법이 있다. 이것은 EU 내에 유용한 데이터 블록이 적은 EU 를 지울 대상으로 선정하는 방식으로 Cost-benefit 방법을 사용하여 비용에 비해 이익이 많은 EU 를 선택해서 지우는 방법이다. 또한 등급별 지움 정책을 제안하여, 플래시 메모리를 지우는 횟수를 줄이고 플래시 메모리 내에서 지우는 곳을 고르게 안배함으로써 플래시 메모리의 수명을 연장 할 수 있도록 지움 정책을 설계하여야 한다. 이렇게 플래시 메모리에 저장되는 공간을 고르게 분배하는 것을 Wear-leveling 이라 한다.

2.3 커널 디바이스 드라이버

리눅스는 단일 커널이다. 즉 커널의 모든 기능적인 요소들이 자신의 내부 자료구조와 모든 함수들에 접근할 수 있는 하나의 거대한 프로그램이다.

운영체제 설계의 다른 방법으로는 커널의 각 기능적인 부분들이 별도의 단위로 쪼개지고, 그 사이에 엄격한 통신 매커니즘으로 연결되는 마이크로커널(micro-kernel) 구조가 있다. 이는 시간이 소모되는 프로세스가 아닌 환경 설정 프로세스를 통하여 새로운 컴포넌트를 커널에 추가할 수 있게 한다. 가령 사용자

가 NCR 810 SCSI 용 드라이버를 사용하려고 하는데 이것이 커널에 포함되어 있지 않다고 하자. 그러면 커널의 설정을 바꾸고 다시 컴파일 해야 NCR 810 SCSI 를 사용할 수 있게 될 것이다. 그러나 여기에 다른 대안이 있다.

리눅스는 운영체제를 구성하는 컴포넌트들을 필요로 할 때 동적으로 로드 또는 언로드할 수 있게 한다. 리눅스 모듈은 시스템이 부팅된 후 언제라도 커널에 동적으로 링크될 수 있는 코드 덩어리이다. 또한 모듈이 더 이상 필요하지 않을 때는 커널과의 연결을 해제하고 제거할 수 있다. 리눅스 커널은 디바이스 드라이버와, 네트워크 드라이버나 파일시스템 같은 유사 디바이스 드라이버(pseudo device driver)로 구성되어 있다.

사용자는 insmod 나 rmmod 같은 명령으로 리눅스 커널 모듈을 명확하게 로드 또는 언로드 할 수 있다.

로드된 커널 모듈은 다른 보통 커널 코드처럼 커널의 한 부분이 된다. 모듈은 커널 코드와 똑 같은 권한과 책임을 진다. 즉, 리눅스 커널 모듈은 커널 코드나 디바이스 드라이버처럼 커널을 망가뜨릴 수도 있다는 것이다.

모듈이 자신이 필요로 할 때 커널의 자원을 사용할 수 있으려면, 그것이 어디 있는지 찾을 수 있어야 한다. 로드된 모듈이 잘못 만들어진 것이어서 운영체제를 망가트릴 가능성과는 별도로, 다른 위험 가능성이 있다. 만약 지금 실행하고 있는 커널보다 이전 버전이나 이후 버전 용으로 컴파일 된 모듈을 로드 하면, 즉 모듈이 커널 루틴을 호출할 때 잘못된 인자를 넘겨준다면 문제가 생길 수 있을 것이다. 커널은 모듈을 로드할 때 엄격한 버전 검사를 하여 이런 문제를 선택적으로 막을 수 있다.

3. 파일 시스템 설계

본 연구는 Real-Time Linux 환경에서 사용하는 파일 시스템으로써, 64 비트 어드레스를 지원하며 익스텐드(1kb-64kb)을 할당의 기본 단위로 사용하려고 한다.

임베드디 저장장치인 플래시 메모리는 용량이 작고 수명이 영구적이지 못하다. 저장되는 파일의 크기도 작은 단위로 구성되어 있기 때문에 저장공간의 효율성이 떨어진다. 이에 저장할 데이터의 크기를 체크하여 크기에 맞게 블록을 생성하고 작일 파일을 공간 효율성을 높이고, 수퍼블록에 사용 빈도를 나타내는 블록(Frequent Block)을 추가하여 블록에 대한 등급을 계산함으로써 지울 데이터를 결정하여 지움정책을 구현하려고 한다. 또한 파일 시스템의 효율을 향상 시키기 위한 캐쉬를 유지하는 것에 등급을 이용하려고 한다.

3.1 실험환경

리눅스를 이식하는 내장형 시스템 보드(Embedded System Board: Target)로서 마이크로프로세서는 인텔 StronARM SA1110 을 사용하였고 플래시 메모리는 인텔 플래시 16M 와 32M 의 SDRAM 을 사용하였다

모니터링과 시리얼 다운로드가 가능한 시리얼 포트가 3 개 있고 네트워크에 연결하기 위한 cs9800 이더넷 컨트롤러를 사용하여 네트워크 및 tftp 기능이 가능하도록 하였다.

3.2 파일 시스템 구성

파일 시스템은 64 비트의 주소를 사용하여 0 – 264-1 까지의 주소 번지를 사용한다. 파일 시스템은 파일 시스템에 관한 정보를 저장하는 Superblock, Block Bitmap, Inode Bitmap, Inode Table, Data Blocks 으로 구성되어 있다.

Super Block	Block Bitmap	Inode Bitmap	Inode Table	Data Blocks

(그림 2) 파일 시스템의 구성

파일 시스템은 고정할당 방법과 동적인 파일 할당 방법이 있다. EXT2 에서는 4KB 당 하나의 Inode 를 할당하므로 4KB 이하의 데이터가 많이 존재할 경우 자원 부족 현상이 일어 날 수 있다.

본 실험에서는 각 자원을 파일 크기에 따라 해당블록에서 재할당 받아 사용하여 효율적으로 저장 자원을 이용하였다. 즉, 파일 초기 생성시 데이터의 크기를 체크하여 블록 크기를 데이터의 크기에 맞게 생성하여 다양한 저장에 대한 요구에 유연하게 반응하고 공간을 효율적으로 사용 할 수 있는 이점이 있다.

3.3 메타데이터

고정 블록을 사용할 경우에 다수의 작은 파일의 저장시 자원 부족 현상이 일어 날 수 있다. 본 실험에서는 블록 사이즈를 저장되는 데이터에 크기에 맞게 재조정하므로 작은 크기의 파일을 저장하는데 있어 효율적으로 저장자원을 활용할 수 있게 하였다.

기존 EXT2 에는 저장장치의 수명이 제한되어 있다는 것이 고려되지 않았다. 이에 수퍼블록에 사용빈도를 나타내는 새로운 블록인 Frequent Block 을 추가하여 파일의 사용 횟수를 기록하여 참조된 횟수를 Wu 의 Cost-benefit 에 적용하여 각 블록에 등급을 계산하여 지울 데이터를 선정함으로써 지움 정책을 구현하려 한다.

3.3.1 메타데이터 처리

파일 시스템에 대한 정보를 나타내는 수퍼블록의 구조는 다음과 같다.

Magic Number	Block Size	Free Block	Free Inode	First Inode	Frequent Block

(그림 3) 수퍼블록의 구성

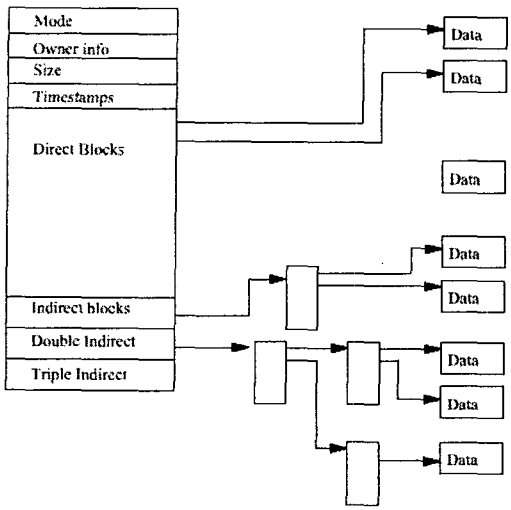
매직넘버는 마운트하는 소프트웨어로 하여금 파일 시스템의 수퍼블록이라는 것을 확인시키고, 블록 사이즈는 시스템의 블록 크기를 바이트 단위로 표시한다. 프리블록은 파일 시스템의 프리 블록 수, 프리 Inode 는 파일 시스템의 프리 Inode 수를 나타낸다. 첫번째 Inode 는 파일 시스템내의 첫번째 Inode 의 Inode 번호를 말한다. Frequeng Block 은 파일의 사용 빈도를 나타

낸다.

3.3.2 고정 Inode 구조

Inode 에 데이터 블록을 위한 포인터를 직접, 간접, 이중, 삼중을 사용하여 고정된 방식의 포인터를 사용한다.

그리고, 디렉토리에 대한 액세스 속도를 높이기 위해, VFS 에서는 디렉토리 엔트리에 대한 캐시를 유지한다.



(그림 4) 고정 Inode 구조

3.3.3 오류 복구

운영 시스템의 비정상적인 동작이나 전원 차단시 파일 시스템에 문제가 발생할 수 있다.

기존 시스템에서는 파일 시스템 체크(fsck)기법을 이용하여 파일 시스템에 발생한 문제점을 해결한다. 이 방법은 파일 시스템의 크기에 비례하여 시간이 증가하고, 오프라인으로 수행된다.

다른 방법인 저널링 기법은 파일 시스템에 오류에 대비해 메타데이터의 정보를 저장하고 오류발생시 저장된 로그를 이용하여 복구하는 방법이다. 이와 같은 로그 파일 기법을 이용하여 변경되는 모든 연산을 로그에 기록한다.

본 연구에서 파일 시스템의 가용성을 높이기 위하여 저널링 기법을 이용하여 파일 시스템에 메타데이터를 변경하는 연산시 이를 기록하여 로그를 만들어 비정상적인 오류가 발생하였을 때 파일 시스템의 일관성을 유지하게 하였다.

3.3.4 파일 시스템 성능 향상

· 파일 시스템의 성능 향상을 위해서 선반입 기법, 버퍼 캐쉬 교체 기법이 있다.

선반입 기법의 경우 사용될 파일을 예상하여 미리 파일을 가져오는 방법이다. 이 방법은 가장 효율적이지만 구현이 어렵다는 문제점이 있다.

버퍼 캐쉬 교체 기법은 버퍼에 가져온 파일을 바로 지우는 것이 아니라 임시로 저장한 다음에 사용 빈도를 예상하여 지우는 것을 결정한다. 이를 결정하기 위하여 메타데이터에 추가한 자료구조인 Frequent Block 에 사용 횟수를 기록함으로써 사용빈도에 대한 등급을 부여할 수 있다. 등급을 계산하기 위하여 Wu 의 Cost-benefit 방법을 이용함으로써 지울 데이터를 선정하여 버퍼를 효율적으로 관리할 수 있었다.

4. 결론 및 향후 연구계획

지금까지 파일 시스템의 특성 및 구조와 플래시 메모리의 특성을 고려하여 리눅스 커널에 파일 시스템을 적용시키기 위한 방법을 살펴보았다. 연구 진행 사항은 EXT2 파일 시스템을 적용한 임베드디 시스템의 메타데이터를 변경하여 적용시키는 작업을 마쳤으며 초기 블록 데이터 크기에 맞는 블록 생성을 올바르게 하고 있는지에 대한 테스트를 하였다.

최근에 임베디드 장비의 저장 매체로 전력소모가 적고 부피가 작은 플래시 메모리를 많이 사용하고 있지만 리눅스 파일 시스템은 회전식 자기 디스크 기반에서 개발되어 플래시 메모리의 특성이 고려되지 않았다. 이에 본 논문에서는 블록 크기를 동적으로 조정하여 저장 공간을 효율적으로 활용하며 파일의 사용 빈도를 나타내는 블록을 추가하여 지움 정책을 구현하였다. 또한 사용 빈도를 나타내는 블록은 파일 시스템의 성능 향상을 위한 버퍼 교체법에 등급정책을 응용하여 적용하고 있다. 오류 복구를 위해서는 저널링 기법을 사용하였다.

향후 연구과제로는 본 파일 시스템에서 고정 inode 를 사용 함으로써 다양한 저장 요구에 유연하게 대처하기 곤란하고, 추가적인 공간 활용이 어려우며 별도의 자료구조 트리를 필요로 하는 문제점을 해결할 방안에 대한 연구가 지속적으로 필요하다.

참고문헌

[1]Alessandro Rubini, "LINUX DEVICE DRIVERS", O'REILLY, 1998

[2]David A Rusling, avid.rusling@arm.com, "The Linux Kernel Copyright © 1996-1999", REVIEW, Version 0.8-3

[3]David A Rusling, "The Linux Kernel", http://linux.flyduck.com/tlk/, 1999.

[4]Moshe Bar, "Linux File System", McGraw-Hill, 2001

[5]Michael J. Folk, Bill Zoellick, "File Structures Second Ediition", Addison-Wesley Publishing Company, 1992

[6]Y.C. Yang and K.J.Lin. "Enhancing the real-time capability of the Linux Kernel", RTCSA'98, Hiroshima, Japan, OCT 1998.

[7]임근수, " The Perfect Analysis of Linux Kernel for Implementing General Purpose Operating System", 2001.

[8]김정기, 박승민, 김채규, "임베디드 플래시 파일 시스템", 정보처리 제 9 권 제 1 호, 2002.1

[9]박성호, 정기동, "내장형 운영체제의 파일 시스템 - Flash File System", 정보처리 제 9 권 제 3 호, 2002.5

[10]이광윤, 정병수, " Embedded System 의 발전 현황" , 삼성 소프트웨어 센터, 2000.