비휘발성 메모리를 위한 파일시스템 설계 및 구현*

백승재[O], 최종무[¶], 이동희[†], 노삼혁[‡]
단국대학교 정보컴퓨터 학부[O¶]
서울시립대학교 컴퓨터 과학부[†]
홍익대학교 정보컴퓨터 공학부[‡]
{ibanez1383, choijm}@dankook.ac.kr, dhlee@venus.uos.ac.kr, samhnoh@hongik.ac.kr

Design and Implementation of a File System for Non-Volatile RAM

Seungjae Baek[O], Jongmoo Choi[¶], Donghee Lee[†], Sam H. Noh[‡]
Division of Information and Computer Science, Dankook University[O¶]
Department of Computer Science, University of Seoul[†]
School of Computer & Information Engineering, Hong-Ik University[‡]

요 약

최근 DRAM 특성인 바이트 단위의 빠른 접근과 디스크나 플래시 메모리 특성인 비휘발성을 동시에 제공하는 차세대 비휘발성 메모리가 등장하고 있다. 본 논문에서는 비휘발성 메모리를 위한 새로운 파일시스템을 제안한다. 이 파일시스템은 메모리 본래의 특성대로 기존의 메모리 공간 할당 함수 인터페이스로 접근이 가능하며, 일반 파일시스템 인터페이스로도 접근이 가능하다. 또한 이 파일시스템은 효율적인 공간관리 및 성능 향상을 위하여 가변 크기 블록 사이즈를 지원한다. 한편 루트 파일시스템 용도로 사용 시 부팅 시간의 단축이 가능하며, page table 매핑 수정을 통해 실행 가능 파일의 직접 수행을 제공한다.

1. 서 론

최근 차세대 비휘발성 메모리 기술의 급격한 발전은 메모리 계층 구조에 대한 변화와 새로운 가능성을 제공하고 있다. 현재 주목 받고 있는 차세대 비휘발성 메모리에는 FRAM(Ferro-electric RAM), MRAM (Magneto-resistive RAM), PRAM(Phase-change RAM) 등이 있다. 최근 이러한 제품에 대한 연구가 수행되어 상용 제품이 출시될 정도로 개발이 진행되었다[1].

이는 기존의 주 기억 장치와 보조 기억 장치가 분리된 메모리 계층 구조에서 통합된 단일 메모리 계층 구조로의 변화를 가능하게 한다. 예를 들면 기존의 DRAM과 디스크를 별도로 사용하는 시스템이 아닌, 차세대 비휘발성 메모리만을 사용하여 영속적인 저장 공간과 프로그램의 동적 수행 공간을 동시에 제공하는 시스템이다. 또한 비휘발성 메모리의 저장 용량이 점점 늘고 있는 추세를 감안하면, 대용량의 메모리에 정보를 체계적으로 저장하고 관리하는 기술은 필수적이라 하겠다.

그러나 현재 대다수의 파일시스템은 기본적으로 메모리가 아닌 보조 저장장치를 위하여 설계되었으며, 소수의 메모리용 파일시스템은 메모리가 휘발성이라는 전제하에 설계되어 있다. 즉, 비휘발성 메모리의 특성을 고려하지 않고 설계되어 있으므로 그 장점을 살리지 못하고 있는 것이다[2].

이에 본 논문에서는 비휘발성 메모리의 특성 및 장점을 고려한 파일시스템을 설계하여 리눅스 상에 구현하였다. 이를 위하여 비휘발성 메모리로는 FRAM을, 구현을 위한 테스트 보드로는 Intel PXA-255칩을 탑재한 EZ-X5 보드를 사용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 비휘발성 메모리를 위한 파일시스템의 설계를 설명한다. 3장에서는 실제 모듈 형태로 제작된 파일시스템의 구현 및 실험 결과를 기술하며, 4장에서는 결론 및 향후 연구 내용을 정리한다.

2. 설 계

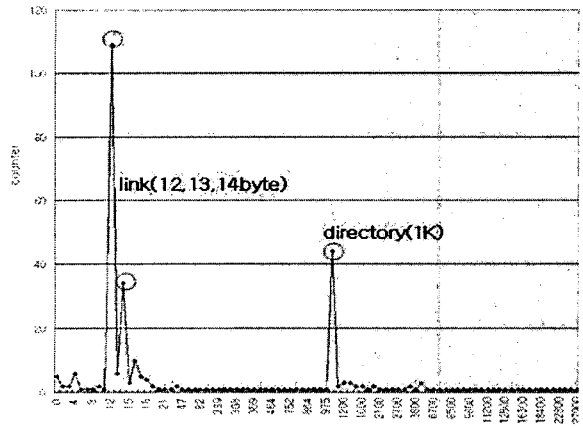
차세대 비휘발성 메모리의 특성과 장점을 최대화하기 위하여 아래와 같이 두 가지 접근 방법을 통해 메모리 공간을 사용할 수 있도록 설계 하였다. 첫째, malloc()등의 메모리 공간 할당 함수 인터페이스로 접근이 가능하다. 할당 받은 공간은 파일 시스템 내에서 Data Block으로 관리되며, Data Block으로 관리되는 공간은 주소를 인자로 하는 별도의 함수(set_file()) 호출을 통해 바로 파일로 변환된다. 둘째, 기존 파일 시스템 인터페이스로도 접근이 가능하다. 즉, open(), read(), write(), close() 등의 일반적인 파일 시스템 인터페이스로 차세대 비휘발성 메모리내의 파일에 접근이 가능함을 말한다. 이러한 특징으로 인해, 파일 연산의 빠른 수행이 가능하다. 즉, malloc()등의 함수를 통해 메모리 공간을 할당 받은 뒤, set_file() 함수를 이용해 이 공간의 데이터를 그대로 파일로 저장하고 관리할 수 있는 구조이다.

파일 시스템 설계에 있어 중요한 요소인 블록 크기를 결정하기 위해 아래와 같은 통계적 방법을 사용하였다. EZ-X5 보드에서 제공하는 루트 파일시스템[3]을 이용하여 임베디드 시스템에서 사용되는 일반적인 루트 파일시스템의 모든 파일 크기와 개수에 대한 통계 정보를 계산하였다.

이 정보는 루트 파일시스템의 최상위 디렉터리에서

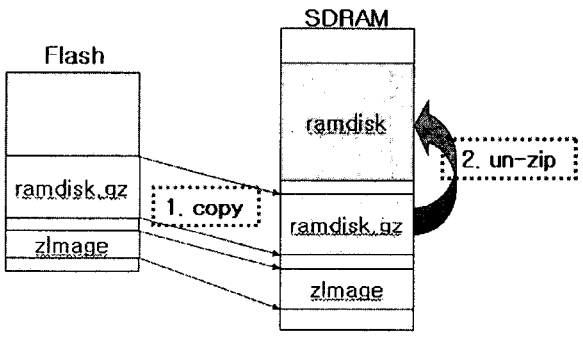
* 본 연구는 한국과학재단 특정기초연구(R01-2004-000-10188-0)지원으로 수행되었음.

"ls -rRhl" 명령을 내려 얻은 결과 중 '/proc'과 같은 파일 시스템에 속한 내용을 모두 제외시킨 뒤 각 파일의 크기를 분석한 것이며, 이를 아래 <그림 1>에 보였다.



<그림 1> root file system내 파일 크기의 정보

<그림 1>의 x축은 파일 크기(byte)를, y축은 파일 개수를 뜻한다. 그림에서 확인 할 수 있듯이 일반적인 루트 파일시스템은 symbolic link에 해당되는 12byte, directory에 해당되는 1K byte크기의 파일이 대다수이다. 이는 고정된 크기의 블록을 사용할 경우 내부 단편화에 따른 성능 저하나 다중 블록 관리를 위한 부하가 있음을 의미하며, 결국 가변 크기 블록 지원의 필요성을 보여준다. 메모리상에 구축되는 파일시스템은 메모리의 기본적인 특성상 디스크 형태의 2차 저장장치와는 달리 물리적으로 연속된 곳에 위치할 필요가 없으며, 이리 고정된 크기의 블록을 사용하여 불필요한 공간낭비를 유발할 이유도 없다. 따라서 본 논문에서는 특정 단위의 블록사이즈를 정하지 않고, 상위레벨에서 요청한 메모리 크기만큼을 할당할 수 있도록 가변크기 블록 사이즈를 지원하는 파일 시스템을 설계하였다.

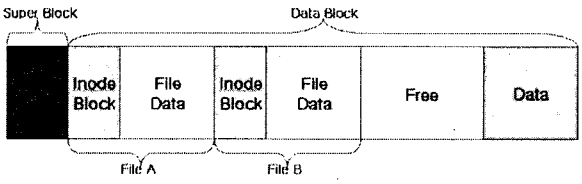


<그림 2> 기존 임베디드 시스템의 부팅

차세대 비휘발성 메모리 기반 파일시스템은 시스템 부팅 시에 성능 향상을 제공할 수 있다. 일반적인 리눅스 기반의 임베디드 시스템은 아래 <그림 2>에서 보여 지듯이 2차 저장 장치에 해당하는 Flash 메모리로부터 부팅에 필요한 파일(zImage, ramdisk)을 SDRAM으로 복사하며(1. copy), 효율적인 메모리 공간관리를 위해 압축되어 있던 루트 파일시스템 이미지(ramdisk.gz)를 압축해제(2. un-zip) 한다[4].

그러나 부팅에 필요한 파일을 본 논문에서 설계한 차세대 비휘발성 메모리 기반의 파일시스템에 직접 저장한 경우 (1.copy)와 (2.un-zip)에 해당되는 과정이 불필요 해짐으로 인해 부팅시간의 단축이라는 성능 향상을 가져온다.

또한 본 논문에서 설계한 파일시스템에 저장 된 실행가능한 파일의 경우, 커널 내부의 page table 수정을 통해 별도의 복사 과정 없이, 바로 수행 될 수 있다. 이를 위해 파일 당 하나씩 존재하는 Inode Block에 Data 영역의 실제 주소 정보를 저장 하였다. 따라서 실행가능 파일을 본 파일시스템에 저장한 후, 이를 수행하는 경우, 기존 파일 시스템과 비교하여 빠르게 프로그램 수행 시작이 가능하다.



<그림 3> 메모리 관리를 위한 파일시스템 구조

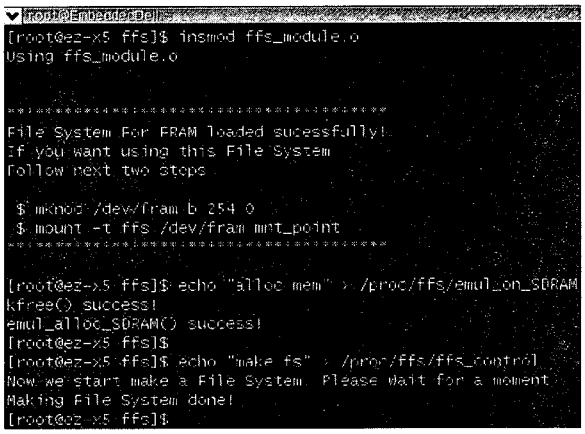
<그림 3>에서 보여 지듯이 파일 시스템 공간은 Super Block 영역과 Data Block 영역으로 나뉜다. Data Block은 다시 파일 당 하나씩 존재하는 고정 크기의 Inode Block과 실제 파일의 내용을 저장하는 File Data 영역, 사용되지 않은 Free 영역으로 나뉜다. Super Block과 Inode Block만 고정된 크기를 가지며, 나머지 영역은 모두 가변 블록 크기를 가진다. Inode Block 없이 사용되고 있는 가장 우측의 Data영역은 메모리 인터페이스로 접근되고 있는 공간을 나타내며 set_file()이 호출되면 Inode Block이 생기고, 그 이후에는 파일시스템 인터페이스로도 접근될 수 있다.

3. 구현 / 실험 결과

본 논문에서는 구현을 위해 모듈 형태로 파일시스템을 구현하였다. 또한 FRAM 접근 루틴과 FRAM이 없는 상태에서도 실험이 가능하도록 SDRAM 상에서 emulating 할 수 있는 루틴을 추가하였다. 현재 본 연구진은

EZ-X5 임베디드 보드에 FRAM을 통합하고 있으며, 아래 실험 결과는 FRAM이 아닌 SDRAM 상에서 emulation 한 결과이다.

아래 <그림 4>는 상기한 바와 같이, SDRAM 상에서 emulating 할 수 있도록 모듈을 컴파일한 뒤의 수행과정을 보인다.



<그림 4> 파일 시스템 초기화

<그림 4>는 파일시스템 초기화 과정을 보여준다. 제안된 파일시스템은 ffs_module.o라는 이름으로 구현되었으며, 모듈을 삽입하면 /proc/ffs 디렉터리 내의 파일로 접근 된다. 우선 SDRAM 상에 FRAM처럼 쓸 수 있도록 임의의 공간을 확보하기 위해 /proc/ffs 디렉터리 내의 파일에게 "alloc mem" 영령을 내린다. 이후 ext2의 'mke2fs'와 유사하게, 파일시스템 초기 구성을 위해 "make fs"명령을 내리면 파일 시스템 사용을 위한 준비가 완료된다.

<그림 5> malloc()로 메모리 할당 및 파일로 저장

다음으로, "(1)메모리 할당, (2)할당받은 메모리에 임의의 정보 기입, (3)set_file()호출"의 순서로 작업을 하는 모듈 'test_drv.o'를 삽입한다. 위 <그림 5>에서 볼 수 있듯이 insmod 명령을 사용하여 모듈을 삽입하면 메모리 할당 후 할당 받은 메모리에 임의의 정보를 써넣고 파일로 만드는 것을 확인 할 수 있다.

마지막으로 현재 생성되어 있는 모든 파일의 이름과 그 내용을 보여주는 역할을 하는 'file view' 명령을 내리면 <그림 5> 하단부에서 볼 수 있듯이 그 정보를 파일로서 유지하고 있음을 확인 할 수 있다.

이는 상기한 본 파일시스템의 장점인, 일련의 메모리 할당 함수를 통해 작업 중이던 공간은 언제든 파일로써 파일 시스템에 의해 관리 될 수 있으며, 또한 본래 목적대로 메모리의 한 영역으로 사용될 수도 있음을 의미한다.

4. 결 론

본 연구에서는 차세대 비휘발성 메모리로 주목을 받고 있는 FRAM을 위한 파일시스템을 설계하여, Intel PXA-255 칩이 내장된 보드를 사용하여 리눅스 상에 구현 하였다. 이는 대용량의 메모리에 정보를 체계적으로 저장 및 관리하는 기술에 대한 요구가 증대하고 있는 요구에 부응하는 기반 기술임에 큰 의미가 있다.

현재, 파일 시스템에 대한 일반적인 접근 함수 보완, 저장되어 있는 파일이 실행가능 한 파일인 경우 page table 매핑을 통한 직접 수행 기능 추가, 기존 루트 파일 시스템을 대체할 수 있도록 보완하는 연구를 진행 중이며, FRAM이 현재의 루트 파일시스템을 대체 할 수 있을 용량으로 발전 되는 것을 고려하여 이에 대한 연구를 진행함을 목표로 한다.

참고문헌

[1] S. Shiratake et al., "A 32Mb chain FeRAM with segment/stitch array architecture", IEEE International Solid-State Circuits Conference, vol.1, p282~283, 2003

[2] Nathan K. Edel et al., "MRAMFS: A compressing file system for non-volatile RAM", MASCOTS'04, 2004

[3] FALINUX, "http://falinux.com"

[4] Daniel P. Bovet et al., Understanding Linux Kernel 2nd edition, p90~93, 2003