임베디드 시스템에서 정적 파일을 활용한 YAFFS2 파일 시스템의 최적화

박병훈*, 서대화*
*경북대학교 전자전기컴퓨터학부
e-mail : dna01@ee.knu.ac.kr

YAFFS2 Filesystem Optimization Using Static Files in Embedded Systems

Byung-Hun Park*, Dae-Wha Seo*
*School of Electriccal Eng. & Computer Science, Kyungpook National University

요 약

플래시 메모리는 현재 다양한 분야에서 전통적인 하드디스크를 점차 대신하고 있다. 이에 따라서 플래시 메모리에 사용될 파일시스템에도 많은 연구가 이루어 지고 있다. 플래시 메모리, 특히 NAND 플래시에 사용되는 가장 대표적인 파일시스템으로 JFFS 와 YAFFS 를 들 수 있는데 YAFFS 가 JFFS 보다 전반적으로 우수하다고 하나 마운트 과정 및 메모리 사용에 있어 비효율적인 면이 존재한다. 본 논문에서는 임베디드 시스템에서 정적 파일을 활용해 YAFFS2 파일시스템의 마운트 시간 및 메모리 사용량을 줄이는 방법을 소개한다.

1. 소개

플래시 메모리는 하드디스크와는 달리 반도체소자로 만들어 지기 때문에 저전력, 내구성, 소형화 등의 특징을 가지고 있다. 이러한 특징은 특히 모바일 임베디드 시스템에 적합하여 휴대폰, PMP 등과 같은 제품에 메인 스토리지로 널리 사용되고 있다.

플래시 메모리는 데이터를 쓰기 전에 지워야 하는 과정이 반드시 필요하고 특히 NAND 플래시의 경우 지우기 단위와 일기/쓰기 단위의 크기가 다르기 때문에 하드디스크에서 사용하던 파일시스템을 그대로 사용할 수가 없다. 따라서 JFFS, YAFFS 등과 같은 플래시 메모리 전용의 파일시스템이나 일반적인 블록디바이스용 파일시스템 하위에 놓이는 FTL 과 같은 특수한 소프트웨어 변환계층이 필요하다.

현재 YAFFS 파일 시스템이 NAND Flash 파일시스템으로 가장 널리 사용이 되고 있다. 하지만 마운트 시 플래시 메모리의 모든 페이지(Page)들을 스캔(Scan)해야 하는 특징 때문에 많은 시간을 필요로 한다. 따라서 YAFFS 를 루트(Root) 파일 시스템으로 사용 할 경우 부트(Boot) 시간도 오래 걸리게 된다. 또한 빠른 페이지 검색을 위해 메모리에 복잡한 데이터 구조를 만들기 때문에 많은 메모리를 요구하게 된다.

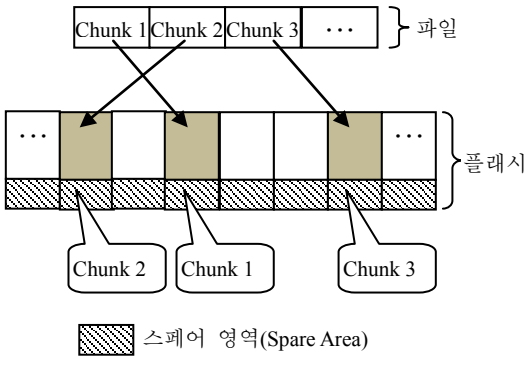
본 논문에서는 임베디드 시스템에서 정적 파일을 활용하여 YAFFS 파일시스템의 마운트 시간 및 메모리 사용량을 줄일 수 있는 방법을 제안한다.

2. 관련 연구

2.1. YAFFS2 파일시스템

2.1.1. 플래시상의 데이터 구조

YAFFS2 파일시스템에서 파일들은 Chunk 단위(플래시의 페이지와 동일한 크기)로 나뉘어 플래시의 각 페이지에 저장이 된다. 페이지의 스페어 영역(Spare Area)에는 그 페이지에 저장된 파일의 Chunk 를 고유하게 식별하기 위해 Object-ID 및 Chunk-ID 가 기록된다[2]. Object-ID 는 일반 파일, 디렉토리, 링크파일 각각의 인스턴스(Instance)를 식별하며 Chunk-ID 로 해당 파일 내에서의 Chunk 을 식별한다(그림 1).



(그림 1) Chunk-ID 의 적용 예

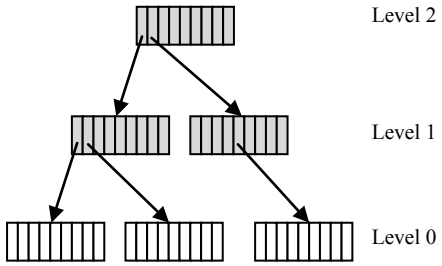
특히 Chunk-ID 0 의 경우 파일 이름, 파일 종류, 파일 길이, 수정 시간 등의 해당 파일의 메타 데이터가 저장이 되고 Chunk-ID 1 번부터 파일의 일반 데이터가 저장된다. 따라서 디렉토리, 링크 Object 의 경우 Chunk-ID 는 0 밖에 존재하지 않는다.

이와 같이 파일의 Chunk-페이지 맵핑(Mapping) 관계

가 플래시상의 특정한 영역에 모여 있지 않고 여러 페이지의 스페어 영역에 흩어져 서로 독립적으로 존재한다. 따라서 파일의 Chunk-페이지 맵핑(Mapping) 관계를 알기 위해서는 마운트 시 플래시상의 모든 페이지를 스캔해야 한다.

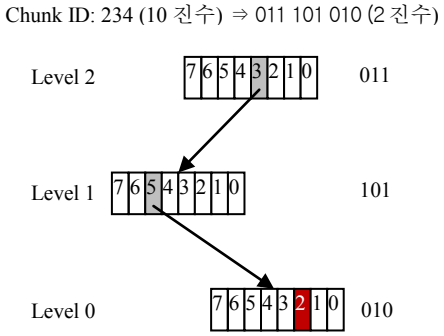
2.1.2. Chunk-페이지 맵핑 구조

YAFFS2 파일시스템은 파일의 Chunk-페이지 맵핑을 위해 메모리상에 각 파일마다 Tnode 트리 구조를 생성한다. 모든 파일의 Tnode 트리는 플래시의 마운트 시 전체 페이지를 스캔하는 과정에서 만들어 진다[1].



(그림 2) Tnode 트리 구조

Tnode 트리의 각 노드들은 배열로 구성되어 있으며 Level 0 에 위치한 노드들의 각 항목에는 파일내의 해당 Chunk 가 위치한 플래시상의 페이지의 주소가 저장된다(그림 2). 그림 3 은 파일의 Tnode 트리와 Chunk-ID 를 사용하여 해당 Chunk 가 저장된 페이지의 주소를 얻는 과정을 나타낸다[1].



(그림 3) Chunk 의 페이지주소 찾기의 예

그림 2,3 에서 파일의 길이가 늘어남에 따라 Tnode 트리의 노드 수가 증가하게 되며 이로 인해 메모리 사용량도 증가하게 된다.

2.1.3. Wear-Leveling

YAFFS2 파일시스템은 특별한 Wear-Leveling 을 수행하지 않는다[1]. 즉, 어떤 블럭에 수정을 필요치 않는 정적 파일들만으로 가득 차 있다면 그 블럭은 결코 변경되는 일이 없다.

2.2. 임베디드 시스템에서의 파일 특징

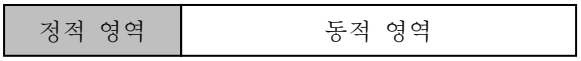
임베디드 시스템은 범용 컴퓨터 시스템과는 달리 특정한 목적을 위해 설계되어 있다. 따라서 대부분의 임베디드 시스템에서 사용되는 파일들은 종류가 다양하지 않으며 그 중에서도 일부만이 수정을 필요로 한다. 특히 임베디드 시스템의 플래시에 초기에 다운로드되는 파일들은 대부분이 수정 및 삭제가 필요치 않고 초기 상태로 존재하는 정적 파일이다.

3. 제안 방법

기본 아이디어는 플래시의 초기 YAFFS2 파일시스템을 구성하는 파일들 중에서 수정이 필요치 않는 정적 파일들만으로 구성되는 플래시상의 영역을 만들어 마운트 시간 및 메모리 공간을 절약하는 방법이다.

3.1. YAFFS2 파일시스템 이미지 생성

임베디드 시스템의 플래시에 초기 YAFFS2 파일시스템을 구성하기 위해 호스트(Host) 컴퓨터에서 필요한 모든 파일이 포함된 파일시스템 이미지(Image)를 생성하여 타깃 보드(Target Board)로 다운로드하는 과정을 거치게 된다. 여기서 이미지를 생성할 때 정적인 파일들(예를 들면 실행파일, 동영상파일 등)을 따로 모아 그림 4 에서와 같이 이미지의 앞쪽에 위치시킨다. 이때 중요한 점은 이미지상에서 정적 영역의 크기는 반드시 플래시 블럭 사이즈의 배수여야 한다. 그래야만 한 블럭에 정적 파일과 동적 파일이 함께 저장이 되는 경우가 발생하지 않기 때문이다.



(그림 4) 파일시스템 이미지 구조

이렇게 생성된 이미지를 플래시에 다운로드하게 되면 플래시 자체도 자연스럽게 정적 영역과 동적 영역으로 나뉘게 된다. 이때 다운로드하고 남은 플래시 영역도 동적 영역에 포함된다.

3.2. Chunk-페이지 맵핑 구조

정적 영역에 존재하는 파일들은 연속된 페이지공간에 저장되며 다른 페이지로 결코 이동이 되는 경우가 없으므로 기존의 Tnode 트리와 같은 복잡한 데이터 구조가 필요하지 않는다. 단지 파일의 첫 Chunk, 즉, Chunk-ID 0 가 위치한 페이지 주소와 파일의 길이만 알면 파일내의 임이의 Chunk 의 위치를 아래의 식을 이용해 알아낼 수 있다.

$$\text{페이지 주소} = \text{Chunk 0 의 페이지 주소} + \text{Chunk Offset}$$

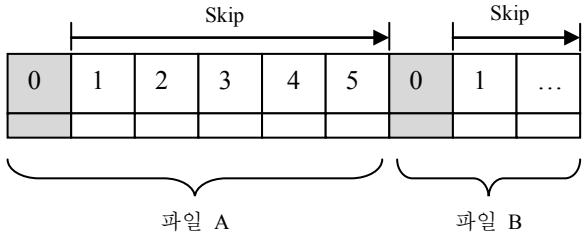
따라서 정적 영역에 대해서는 기존의 Tnode 구조를 만들 필요가 없기 때문에 메모리 공간이 절약되는 효과를 얻을 수 있다.

3.3. 마운트 방법

파일시스템 이미지를 플래시에 다운로드 했을 때 정정 영역 이외의 공간은 모두 동적 영역이 된다. 그리고 이 영역은 시간이 흐름에 따라 수정 및 Garbage

Collection 등의 영향으로 파일의 Chunk 들이 페이지들 상에 흩어지기 된다. 따라서 동적 영역은 기존의 방식과 같이 모든 페이지들의 스캔이 필요하다.

정적 영역의 경우 파일의 Chunk 들이 페이지에 연속적으로 저장되고 이후 변경이 되지 않기 때문에 첫 번째 Chunk, 즉 Chunk-ID 0 에 저장된 파일의 길이를 알아내면 이후 마지막 Chunk 가 위치한 페이지까지 스캔없이 건너 뛸 수 있다(그림 5).



(그림 5) 정적 영역의 스캐닝

따라서 스캔과정에서 건너뛰는 페이지의 수가 많을수록 마운트 시간이 감소됨을 알 수 있다. 단축되는 시간을 구하는 식은 그림 6 에서와 같이 유도해 낼 수 있다.

그림 6 에 나타난 식을 사용하면 기존 방법의 마운트 시간($_{ori}$MountTime)과 정적 영역의 블록 수($_{static}$BlockNum) 및 파일 수($_{static}$FileNum)를 알면 제안하는 방법의 마운트 시간($_{new}$MountTime)을 구할 수 있음을 알 수 있다. 또한 정적 영역의 블럭 수는 많을수록 파일 수는 적을수록 마운트 시간이 줄어드는 것을 알 수 있다.

4. 결론

본 논문에서는 임베디드 시스템에서 사용되는 파일들의 대부분이 정적 파일임에 착안하여 이를 이용하여 YAFFS2 파일시스템에서 마운트 시간 및 메모리 공간을 줄이는 방법을 제안했다. 이 방법은 정적 파일들이 수가 적고 각각의 길이가 길수록 유리하기 때문에 고용량의 다수의 미디어 데이터을 변경 없이 계속 반복 재생하는 임베디드 멀티미디어 시스템 같은 곳에 적합하다 할 수 있다.

$_{ori}$MountTime = (($_{total}$BlockNum – $_{free}$BlockNum) * PagePerBlock) * $_{spare}$ScanTime

$_{new}$MountTime = $_{static}$Time + $_{dynamic}$Time
$_{static}$Time = $_{static}$FileNum * $_{spare}$ScanTime
$_{dynamic}$Time = (($_{total}$BlockNum – $_{static}$BlockNum – $_{free}$BlockNum) * PagePerBlock) * $_{spare}$ScanTime
$_{new}$MountTime = ($_{static}$FileNum + ($_{total}$BlockNum – $_{static}$BlockNum – $_{free}$BlockNum) * PagePerBlock) * $_{spare}$ScanTime

$_{ori}$MountTime – $_{new}$MountTime = ($_{static}$BlockNum * PagePerBlock – $_{static}$FileNum) * $_{spare}$ScanTime

$_{ori}$MountTime	: 기존의 방법으로 했을 때의 플래시 마운트 시간
$_{new}$MountTime	: 제안된 새로운 방법으로 했을 때의 플래시 마운트 시간
$_{total}$BlockNum	: 플래시의 총 블럭 수
$_{free}$BlockNum	: 플래시에서 사용되지 않은 순수한 블럭(Clean Block)
$_{static}$BlockNum	: 플래시에서 정적 영역이 차지하는 블럭 수
$_{static}$Time	: 플래시에서 정적 영역을 스캔하는데 걸리는 시간
$_{dynamic}$Time	: 플래시에서 동적 영역을 스캔하는데 걸리는 시간
PagePerBlock	: 플래시의 블럭당 페이지 수
$_{spare}$ScanTime	: 플래시에서 페이지의 스페어 영역을 스캔하는데 걸리는 시간
$_{static}$FileNum	: 플래시에서 정적 영역을 차지하고 있는 정적 파일의 수

(그림 6) 제안방법의 단축시간을 구하는 식

참고문헌

[1] YAFFS Spec, http://www.yaffs.net/yaffs-spec
[2] YAFFS2 Specification and Development Notes, http://www.yaffs.net/yaffs-2-specification-and-development-notes