NAND 플래시 파일 시스템을 위한 안전 삭제

이재흥° 오진하 김석현 이상호 허준영 조유근 홍지만*
서울대학교 컴퓨터공학부
숭실대학교 컴퓨터학부*
{jhlee°, jhoh, shkim, shyi, jyheo, ykcho}@ssrnet.snu.ac.kr jiman@ssu.ac.kr

Secure Deletion for NAND Flash File System

Jaeheung Lee° Jinha Oh Seokhyun Kim Sangho Yi Junyoung Heo Yookun Cho
Jiman Hong*
Seoul National University Soongsil University*

대부분의 파일 시스템에서는 파일이 삭제되더라도 메타데이터만 삭제되거나 변경될 뿐, 파일 데이터는 물리 매체에 계속 존재하게 된다. 그렇기 때문에 지워진 파일을 복구하는 것도 가능한데 일부 사용자는 이러한 복구를 불가능하게 하는 것을 바라기도 한다. 이러한 요구는 플래시 메모리를 저장 매체로 사용하는 임베디드 시스템에서 더욱더 많아지고 있다. 이에 본 논문에서는 YAFFS를 변경하여 안전삭제 기능을 가진 NAND 플래시 파일시스템을 설계하였다. 제시한 기법은 암호화 키 뿐만 아니라 파일의 메타데이터까지 안전하게 삭제한다.

제안 기법은 파일을 안전하게 삭제하기 위해 암호화를 이용하는 데, 모든 파일은 파일마다 다른 키로 암호화된다. 이 키들은 파일이 생성되거나 변경될 때 임의로 생성된다. 파일의 데이터를 암호화하는데 사용된 키는 파일의 메타데이터를 저장하는 부분인 객체 헤더에 저장된다. 그러므로 파일의 키를 저장한 객체 헤더가 삭제된다면 파일의 데이터를 복구하는 것도 불가능해진다. 그러나 YAFFS2는 로그 기반 구조로 이루어져 있으므로 파일 시스템 안에 동일한 파일 ID와 chunk 번호를 가지는 페이지들이 여러 개 있을 수 있다. 그러므로 파일의 데이터를 복구 불가능하게 만들기 위해서는 파일의 현재 객체 헤더 뿐만 아니라 이전의 생성되었던 모든 객체 헤더들도 삭제되어야 한다. 우리가 제시한 기법은 한 특정 파일의 현재 및 과거의 객체 헤더들이 모두 하나의 블록에 저장되도록 한다. 그래서 단 한 번의 지우기 연산으로 안전하게 파일을 삭제하는 것이 가능하다.

플래시 메모리는 고정된 크기를 가진 블록으로 구성된다. 제안 기법은 블록을 헤더 블록과 데이터 블록의 두 가지 영역으로 분리한다. 객체 헤더들은 헤더 블록에 저장되고 파일 데이터들은 데이터 블록에 저장된다. 이러한 블록들은 그 블록의 페이지들의 chunk 번호가 0인지 살핌으로써 구분할 수 있다. chunk 번호가 0이면 그 블록은 헤더 블록이고 그렇지 않으면 그 블록은 데이터 블록이다.

본 논문에서 제안하는 기법은 이진 트리 구조를 사용한다. 이 트리는 DRAM 메모리에서 유지되고, 새로 생성된 객체 헤더가 지정된 헤더 블록에 저장되도록 하는 데에 이용된다. 이 트리 구조에서, 각 노드는 유일한 value를 가지고 루트 노드의 경우 이 value는 NULL이다. 노드의 왼쪽 자식은 부모 노드의 value의 왼쪽에 0을 덧붙인다. 반면에 노드의 오른쪽 자식은 부모 노드의 value의 왼쪽에 1을 덧붙인다. 각 단말 노드는 헤더 블록에 대한 포인터를 가진다. 새로 생성한 객체 헤더를 헤더 블록에 저장하려고 할 때, 파일 ID의 n개의 LSB(Least Significant Bits)가 노드의 깊이가 n인 단말 노드의 value와 같아야 한다. 그래서 같은 파일 ID를 가지는 모든 객체 헤더는 같은 헤더 블록에 저장된다. 본 파일 시스템이 마운트 될 때 이진 트리의 구성을 더 쉽게 하기 위해 객체 헤더 구조에 HB_value와 HB_depth라고 명명한 두 개의 필드를 추가했다. HB_value는 노드의 value이고 HB_depth는 노드 vaule의 길이이다.

그림 1은 이진 트리와 이와 관련된 헤더 블록의 예를 보여준다. 이 그림에서 값이 '0001' 인 노드는 파일 ID가 '0001' 로 끝나는 객체 헤더를 저장하는 헤더 블록을 가리킨다. 그래서 파일 ID가 '0001' 로 끝나는 모든 객체 헤더는 같은 헤더 블록에 저장된다.

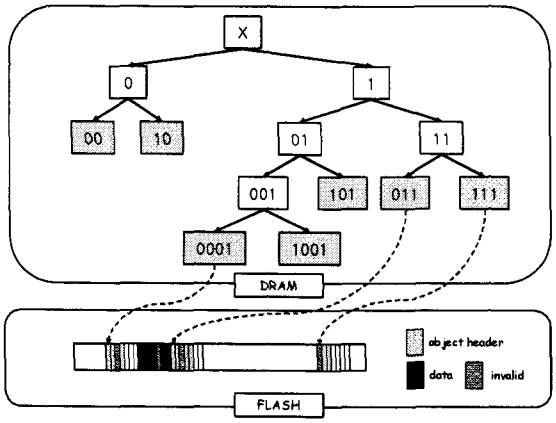

그림 1. 이진 트리와 이와 연결된 헤더 블록의 예

YAFFS에서 각각의 객체 헤더는 하나의 페이지를 차지한다. 한 블록은 32 또는 64개의 페이지로 구성되어 있으므로 각 헤더 블록은 최대 32 또는 64개의 객체 헤더를 담을 수 있다. 그러므로 헤더 블록의 객체 헤더의 수를 한 블록의 페이지 수로 제한해야 할 필요성이 있다. 이를 수행하기 위해 다음과 같은 연산들을 정의하였다.

[OBJECT_HEADER_ADD 연산]

(1) 생성하거나 변경하려는 파일의 ID를 얻는다.
(2) 노드의 깊이가 n이고, 파일 ID의 n개의 LSB와 같은 Value를 가지는 단말 노드를 탐색한다. 이것은 루트 노드에서 시작하고, 단말 노드가 발견될 때까지 파일 ID의 LSB값의 순서대로 트리의 노드를 따라간다.
(3) (2)에서 발견된 단말 노드가 가리키는 헤더 블록에 새로운 객체 헤더가 들어갈 공간이 있는지 확인한다. 빈 공간이 있다면 객체 헤더를 그 헤더 블록에 저장하고 연산을 끝낸다.
(4) 만약 빈 공간이 없다면 그 헤더 블록을 DRAM에 복사한다.
(5) 헤더 블록의 유효한 객체 헤더의 수를 센다.
(6) 그 헤더 블록의 유효한 객체 헤더의 수가 한 블록의 페이지 수의 반을 넘으면, 이전의 헤더 블록의 value에 0과 1을 각각 덧붙인 value를 가진 두 개의 헤더 블록을 할당하고, 유효한 객체 헤더와 생성된 객체 헤더를 그들의 파일 ID에 따라 두 개의 헤더 블록에 나누어 저장한다.
(7) 만약 한 블록의 페이지수의 반을 넘지 않으면 이전의 헤더 블록의 value와 동일한 value를 가진 하나의 헤더 블록을 할당하고 이 블록에 기존의 유효한 객체 헤더와 생성된 객체 헤더를 저장한다.
(8) (4)에서 복사된 블록을 지운다.

[OBJECT_HEADER_DEL 연산]

(1) 안전삭제를 하려는 파일의 ID를 얻는다.
(2) 노드의 깊이가 n이고, 파일 ID의 n개의 LSB와 같은 value를 가지는 단말 노드를 탐색한다. 이것은 루트 노드에서 시작하고, 단말 노드가 발견될 때까지 파일 ID의 LSB값의 순서대로 트리의 노드를 따라간다.
(3) (2)에서 발견된 단말 노드가 가리키는 헤더 블록을 DRAM으로 복사한다.
(4) (1)에서 발견된 파일 ID를 가진 객체 헤더가 유효하지 않는 것으로 설정 한다.
(5) 이전의 헤더 블록의 값과 같은 값을 가지는 하나의 헤더 블록을 할당하고 모든 유효한 객체 헤더를 할당된 블록에 저장한다.
(6) (3)에서 복사된 블록을 지운다.