(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2016-0012769
(43) 공개일자 2016년02월03일

(51) 국제특허분류(Int. Cl.)
G06F 12/02 (2006.01) *G06F 9/44* (2006.01)
(21) 출원번호 10-2014-0094911
(22) 출원일자 2014년07월25일
심사청구일자 2014년07월25일

(71) 출원인
한림대학교 산학협력단
강원도 춘천시 한림대학길 1, 한림대학교(옥천동)
(72) 발명자
고영웅
강원도 춘천시 춘주로 174, 107동 1501호 (퇴계동,그린타운아파트)
이선우
강원도 춘천시 춘주로 174, 104동 505호 (퇴계동,그린타운아파트)
송창근
강원도 춘천시 춘주로 174, 102동 1505호 (퇴계동,그린타운아파트)
(74) 대리인
특허법인세신

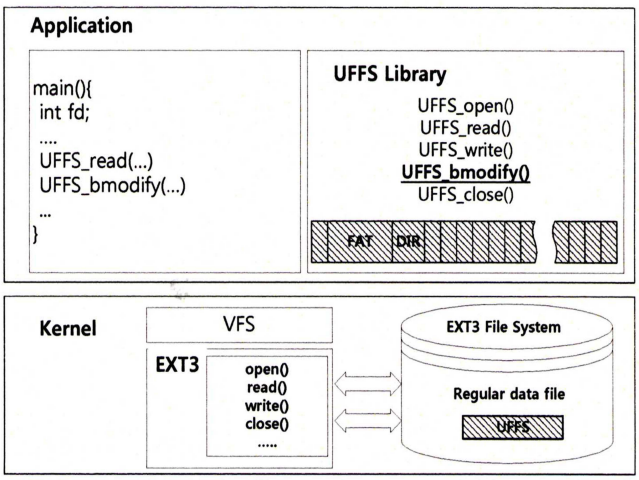
전체 청구항 수 : 총 8 항

(54) 발명의 명칭 **파일 할당 테이블 파일 시스템 및 파일 저장 방법**

(57) 요 약

본 발명은 파일 할당 테이블 파일 시스템 및 파일 저장 방법에 관한 것으로, 본 발명에 따른 파일 할당 테이블 파일 시스템은, 부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 하나 이상의 파일을 저장하기 위한 정보 저장 모듈과, 애플리케이션 프로그램이 구동되면 정보 저장 모듈에 저장된 해당 파일을 읽어오고, 애플리케이션 프로그램에 따라 파일이 편집된 경우 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 정보 저장 모듈에 편집된 파일을 저장하게 하고, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 정보 저장 모듈에 편집된 파일을 저장하게 하는 애플리케이션 모듈을 포함함으로써, 블록 단위의 파일 수정을 빠르게 수행하게 할 수 있다.

대 표 도 - 도3



명 세 서

청구범위

청구항 1

파일 할당 테이블 파일 시스템에 있어서,

부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 하나 이상의 파일을 저장하기 위한 정보 저장 모듈과, 및

애플리케이션 프로그램이 구동되면 상기 정보 저장 모듈에 저장된 해당 파일을 읽어오고, 애플리케이션 프로그램에 따라 파일이 편집된 경우 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하고, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 애플리케이션 모듈을 포함하는 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 2

제1항에 있어서,

상기 애플리케이션 모듈은 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 기록 파일 할당 테이블 처리부와, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 편집 할당 테이블 처리부를 포함하는 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 3

제2항에 있어서,

상기 기록 파일 할당 처리부에서 제공되는 제어 요소들은 파일명, 주소 및 크기이며, 상기 편집 파일 할당 처리부에서 제공되는 제어 요소들은 파일명, 주소, 오프셋, 크기 및 옵션을 포함하고, 상기 옵션은 블록이 추가된 경우와 블록이 삭제된 경우를 나타내는 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 4

제1항 내지 제3항 중 어느 한 항에 있어서,

상기 편집 라이브러리는 블록이 추가된 경우 추가된 블록의 개수만큼 상기 정보 저장 모듈의 데이터 영역 중 대응하는 개수의 클러스터들에 데이터를 저장하고 추가된 블록을 위해 파일 할당 테이블 영역의 파일 할당 테이블을 링크시키는 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 5

제4항에 있어서,

상기 편집 라이브러리는 블록이 삭제된 경우 상기 정보 저장 모듈의 데이터 영역으로의 기록 없이 삭제된 블록에 해당하는 파일 할당 테이블 영역의 파일 할당 테이블의 엔트리를 제로로 변경하고 삭제된 블록을 위해 파일 할당 테이블을 링크시키는 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 6

제1항 내지 제3항 중 어느 한 항에 있어서,

상기 정보 저장 모듈의 저장 매체는 반도체 타입의 불휘발성 메모리인 것을 특징으로 하는 파일 할당 테이블 파일 시스템.

청구항 7

부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 하나 이상의 파일을 저장하기 위한 정보 저장 모듈을 포함하는 파일 할당 테이블 파일 시스템의 파일 할당 테이블 파일 저장 방법에 있어서,

애플리케이션 프로그램이 구동되면 상기 정보 저장 모듈에 저장된 해당 파일을 읽어오는 파일 독출 단계와,

상기 애플리케이션 프로그램에 따라 파일을 편집하는 파일 편집 단계와, 및

상기 파일 편집 단계에서 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 상기 편집된 파일을 저장하고, 상기 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 상기 편집된 파일을 저장하는 파일 저장 단계를 포함하는 것을 특징으로 하는 파일 할당 테이블 파일 저장 방법.

청구항 8

제7항에 있어서,

상기 파일 저장 단계에서, 상기 편집 라이브러리는 블록이 추가된 경우에는 추가된 블록의 개수만큼 상기 정보 저장 모듈의 데이터 영역 중 대응하는 개수의 클러스터들에 데이터를 저장하고 추가된 블록을 위해 파일 할당 테이블 영역의 파일 할당 테이블을 링크시키고, 블록이 삭제된 경우에는 상기 정보 저장 모듈의 데이터 영역으로의 기록 없이 삭제된 블록에 해당하는 파일 할당 테이블 영역의 파일 할당 테이블의 엔트리를 제로로 변경하고 삭제된 블록을 위해 파일 할당 테이블을 링크시키는 것을 특징으로 하는 파일 할당 테이블 파일 저장 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 파일 할당 테이블 파일 시스템 및 파일 저장 방법에 관한 것으로, 특히 블록 단위의 파일의 수정을 용이하게 할 수 있는 파일 할당 테이블 파일 시스템 및 파일 저장 방법에 관한 것이다.

배 경 기 술

[0002] 블록 데이터 입출력을 이용하는 애플리케이션 프로그램들의 예들은 다음과 같다. 첫 번째로, Winzip 압축 툴들에 있어서, 서브 파일들은 하나의 압축 파일 내에 결부된다. 두 번째로, TAR 기록 파일은 많은 서브-파일들을 포함하며, 또한 서브-파일 삽입 및 삭제를 겪게 된다. 세 번째로, 서브-파일 관리 방식은 또한 멀티미디어 편집에 유용한데, 디지털 비디오 편집에 있어서, 그것은 어떤 프레임에도 액세스할 수 있으며, 컷-앤드-페이스트(데이터의 일부를 떼어 이동 삽입하기) 방법을 사용할 수 있다.

[0003] 하지만, 지금까지 사용자가 이들 파일에 서브-파일을 추가하거나 제거하고자 한다면, 일반적으로 올드 파일에 해당하는 전체 데이터 블록들을 다시 써야하므로 매우 많은 디스크 입출력을 발생시킨다. 예를 들어, 40K 바이트를 갖는 파일과 파일 시스템의 블록 크기가 4K 바이트라고 가정하면, 파일에 저장될 수 있는 데이터 블록들의 전체 수는 10이다. 애플리케이션 프로그램이 파일 위치 0에 4K 바이크 크기를 갖는 데이터 블록을 삽입하는 경우, 애플리케이션 프로그램은 올드 파일의 클러스터들 모두를 오버라이트해야 한다. 비록 올드 파일 상에 동일 블록들이 있을지라도, 파일 시스템은 새로운 파일에 모든 블록들을 다시 쓴다. 이러한 종래의 파일 변경 접근은 파일 변경이 블록 정렬 방식일지라도 파일을 변경하는 경우 매우 많은 디스크 입출력 요청들을 발생시킨다. 하지만, 종래의 파일 시스템은 파일을 스트림 바이트로서 고려하기 때문에 파일에 대한 블록 입출력을 지원하지 않았다.

발명의 내용

해결하려는 과제

[0004] 상술한 문제를 해결하기 위해, 본 발명은 블록 단위의 파일 수정을 용이하게 할 수 있는 파일 할당 테이블 파일 시스템 및 파일 저장 방법을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0005] 상술한 목적을 달성하기 위해, 본 발명의 일실시예에 따른 파일 할당 테이블 파일 시스템은, 부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 하나 이상의 파일을 저장하기 위한 정보 저장 모듈과, 애플리케이션 프로그램이 구동되면 상기 정보 저장 모듈에 저장된 해당 파일을 읽어오고, 애플리케이션 프로그램에 따라 파일이 편집된 경우 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하고, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 애플리케이션 모듈을 제공한다.

[0006] 상기 애플리케이션 모듈은 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 기록 파일 할당 테이블 처리부와, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 편집된 파일을 저장하게 하는 편집 할당 테이블 처리부를 포함할 수 있다.

[0007] 상기 기록 파일 할당 처리부에서 제공되는 제어 요소들은 파일명, 주소 및 크기이며, 상기 편집 파일 할당 처리부에서 제공되는 제어 요소들은 파일명, 주소, 오프셋, 크기 및 옵션을 포함할 수 있고, 상기 옵션은 블록이 추가된 경우와 블록이 삭제된 경우를 나타낼 수 있다.

[0008] 상기 편집 라이브러리는 블록이 추가된 경우 추가된 블록의 개수만큼 상기 정보 저장 모듈의 데이터 영역 중 대응하는 개수의 클러스터들에 데이터를 저장하고 추가된 블록을 위해 파일 할당 테이블 영역의 파일 할당 테이블을 링크시킬 수 있다.

[0009] 상기 편집 라이브러리는 블록이 삭제된 경우 상기 정보 저장 모듈의 데이터 영역으로의 기록 없이 삭제된 블록에 해당하는 파일 할당 테이블 영역의 파일 할당 테이블의 엔트리를 제로로 변경하고 삭제된 블록을 위해 파일 할당 테이블을 링크시킬 수 있다.

[0010] 상기 정보 저장 모듈의 저장 매체는 반도체 타입의 불휘발성 메모리일 수 있다.

[0011] 본 발명의 다른 실시예에 따른, 부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 하나 이상의 파일을 저장하기 위한 정보 저장 모듈을 포함하는 파일 할당 테이블 파일 시스템의 파일 할당 테이블 파일 저장 방법은, 애플리케이션 프로그램이 구동되면 상기 정보 저장 모듈에 저장된 해당 파일을 읽어오는 파일 독출 단계와, 상기 애플리케이션 프로그램에 따라 파일을 편집하는 파일 편집 단계와, 상기 파일 편집 단계에서 편집된 파일이 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 상기 정보 저장 모듈에 상기 편집된 파일을 저장하고, 상기 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 상기 정보 저장 모듈에 상기 편집된 파일을 저장하는 파일 저장 단계를 제공함으로써, 상술한 목적을 달성할 수 있다.

발명의 효과

[0012] 상술한 구성에 의해, 본 발명은 블록 단위의 파일 수정을 빠르게 수행하게 할 수 있다.

[0013] 또한, 본 발명은 블록 단위의 데이터가 삭제된 경우에는 저장 매체의 데이터 영역에 대한 기록을 회피할 수 있다.

[0014] 또한, 본 발명은 저장 매체의 데이터 영역에 대한 기록을 최소화함으로써 반도체 타입의 불휘발성 메모리인 플래시 메모리 등의 수명을 연장할 수 있다.

도면의 간단한 설명

[0015] 도 1a 및 도 1b는 일반적인 FAT 파일 시스템의 구조를 도시하는 도면이다.

도 2는 도 1에 도시된 FAT의 구조를 도시하는 도면이다.

도 3은 본 발명의 일실시예에 따른 사용자 레벨의 파일 시스템의 구조를 도시하는 도면이다.

도 4는 도 3에 도시된 파일 시스템의 파일 수정의 예들을 도시한 도면이다.

도 5는 도 3에 도시된 파일 시스템의 블록도를 도시하는 도면이다.

도 6은 본 발명의 다른 실시예에 따른 파일 저장 방법의 흐름도를 도시하는 도면이다.

도 7은 본 발명에 따른 파일 저장 방법의 서브-파일 관리와 통상의 파일 저장 방법의 서브-파일 관리를 비교한 그래프를 도시하는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0016] 이하, 첨부된 도면을 참조하여 본 발명에 따른 파일 시스템 및 파일 저장 방법의 바람직한 실시예를 설명한다. 참고로, 아래에서 본 발명을 설명함에 있어서, 본 발명의 구성요소를 지칭하는 용어들은 각각의 구성 요소들의 기능을 고려하여 명명된 것이므로, 본 발명의 기술적 구성요소를 한정하는 의미로 이해되어서는 안 될 것이다.

[0017] 도 1a 및 도 1b는 일반적인 FAT 파일 시스템의 구조를 도시하는 도면이다.

[0018] FAT 파일 시스템은 가장 단순한 종류의 파일 시스템들 중 하나이다. 도 1a 및 도 1b에 도시된 바와 같이, FAT 파일 시스템은 파일들과 디렉토리들을 저장하기 위해, 부트 레코드를 포함하는 예약된 영역, 파일 할당 테이블(FAT) 영역, 루트 디렉토리 영역 및 데이터 영역으로 구성된다.

[0019] 도 2는 도 1에 도시된 FAT의 구조를 도시하는 도면이다.

[0020] FAT는 데이터 블록들을 위한 일종의 작은 맵으로, 데이터 영역의 클러스터들을 관리하는 테이블이 모여 있는 공간이다. FAT 영역을 통해 어떤 클러스터가 비어 있는지, 그리고 어떤 파일에 어떤 클러스터가 연결되어 있는지를 알 수 있다. 즉, FAT 엔트리의 제로 값은 블록이 사용되지 않음을 나타내고, FAT 엔트리의 특정 값(0xFFF)은 파일의 종료를 나타내며, FAT의 나머지 엔트리 값은 해당 파일의 다음 클러스터들을 나타낸다.

[0021] 도 2의 예시에서 클러스터 크기는 4K 바이트로 가정한다. 도 2의 예시에서, 파일명은 foo이며, foo 파일의 제1 클러스터 번호는 100이며, 해당 파일은 6개의 클러스터들, 즉 24K 바이트를 사용한다.

[0022] 도 3은 본 발명의 일실시예에 따른 사용자 레벨의 파일 시스템의 구조를 도시하는 도면이고, 도 4는 도 3에 도시된 파일 시스템의 파일 수정의 예들을 도시한 도면이다.

[0023] 도 3에 도시된 사용자 레벨의 FAT 파일 시스템은 FAT 리스트에 엔트리를 삽입하거나 또는 삭제하여 FAT의 클러스터 번호를 변경함으로써 불록을 삭제하거나 또는 삽입할 수 있다.

[0024] 도 3에 도시된 바와 같이, 사용자 레벨의 FAT 파일 시스템 라이브러리들은 통상의 UFFS_open(), UFFS_read(), UFFS_write() 및 UFFS_close() 이외에 새롭게 추가된 UFFS_bmodify()를 포함한다. 여기서, UFFS_open()은 해당 파일을 개방하기 위한 라이브러리이며, UFFS_read()는 해당 파일을 읽어오기 위한 라이브러리이며, UFFS_write()는 해당 파일을 기록하기 위한 라이브러리이며, UFFS_close()는 해당 파일을 폐쇄하기 위한 라이브러리이며, UFFS_bmodify()는 해당 파일이 편집된 경우 편집된 데이터 블록에 대해서만 기록하기 위한 라이브러리이다.

[0025] 그리고 커널에 도시된 가상 파일 시스템(VFS)에서의 각 라이브러리와 관련된 제어 요소들은 다음과 같다.

[0026] open(fn)

[0027] read(fd, buffer, size)

[0028] write(fd, buffer, size)

[0029] bmodify(fd, buffer, offset, size, option)

[0030] 그리고 fd=open(fn)이다.

[0031] 여기서 fn은 파일명이며, fd는 파일 디스크립터, buffer는 파일 주소, size는 파일 크기, offset는 파일 오프셋 그리고 option은 블록의 추가 또는 삭제를 나타낸다.

[0032] 따라서 본 실시예서 추가된 파일 입출력 라이브러리인 bmodify(fd, buffer, offset, size, option)는 오프셋과 오프셋 + 크기 영역 사이의 FAT의 클러스트 번호를 재 링크할 수 있다.

[0033] 본 실시예의 구현예로서는 리눅스 EXT4 파일 시스템에 큰 용량 파일(이 구현에서는 10G 바이트)을 생성하며, 파일을 FAT DIR, 및 데이터 블록들을 포함하는 복수의 영역들로 국부적으로 분할한다. 도 3에 도시된 바와 같이, 사용자 레벨의 FAT 파일 시스템 파티션은 리눅스 파일 시스템 상의 규칙적인 파일이다. 따라서 EXT4 파일 시스템의 일반적인 파일 입출력 동작으로 사용자 레벨의 FAT 파일 시스템을 완전하게 제어할 수 있다. 사용자 레벨의 FAT 파일 시스템의 모든 기능들은 UFFS_bmodify()를 제외하고는 일반적인 파일 시스템 기능들과 매우 유사하

다.

[0034] 예를 들면, 도 4에서 사용자 레벨의 FAT 파일 시스템은 파일 오프셋 4096으로부터 foo 파일 상에 4K 바이트 데이터를 삽입할 수 있다. 이 경우 사용자 레벨의 FAT 파일 시스템은, 첫째로 사용자 레벨의 FAT 파일 시스템으로부터 프리 클러스터(클러스터 번호는 103임)를 얻고, 이어서 클러스터 오프셋으로부터 103에 4K 바이트를 복사한다. 둘째로, 파일 할당 테이블의 100번째 클러스터의 엔트리는 103으로 변경되고, 또한 엔트리를 링크-리스트로 삽입하듯이 103번째 클러스터의 엔트리는 101로 변경된다.

[0035] 한편, 블록 삭제는 데이터 영역으로의 사실상의 데이터 복사 없이 취급될 수 있다. 사용자 레벨의 FAT 파일 시스템이 오프셋 8192 상의 foo 파일로부터 2개의 블록들을 삭제한다면, 사용자 레벨의 FAT 파일 시스템은 해당 클러스터의 엔트리들을 000로 변경하여 마킹함으로써 2개의 엔트리들을 삭제한다.

[0036] 도 4의 예시에서, foo 파일에 블록을 삽입하는 경우에는 디스크에 기록되는 블록은 1블록이며, 블록을 삭제하는 경우에는 디스크에 데이터 블록들을 위한 기록이 생기지 않는다.

[0037] 도 5는 도 3에 도시된 파일 시스템의 블록도를 도시하는 도면이다.

[0038] 도 5에 도시된 바와 같이, 사용자 레벨의 FAT 파일 시스템은 애플리케이션 모듈(510) 및 정보 저장 모듈(550)을 포함한다.

[0039] 애플리케이션 모듈(510)은 CPU(512), RAM(514), 정보 저장 모듈 인터페이스(516) 및 ROM(520)을 포함한다. ROM(520)에는 애플리케이션 모듈(510)을 제어하는 프로그램이 저장되어 있다. 애플리케이션 프로그램이 동작하면, CPU(512)는 ROM(520)에서 이 애플리케이션 프로그램을 읽어와 RAM(514)에 일시적으로 저장한다. 정보 저장 모듈 인터페이스(516)는 정보 저장 모듈(550)과 통신하기 위한 인터페이스로, 제어 신호 및 데이터의 송수신을 수행한다.

[0040] ROM(520)은 애플리케이션 처리부(530), FAT 처리부(532) 및 정보 저장 모듈 액세스부(538)를 포함한다. 애플리케이션 처리부(530)는 데이터의 생성이나 전원의 제어 등 애플리케이션 모듈(510) 전체의 제어를 실시한다. FAT 처리부(532)는 FAT 파일시스템에 의해 데이터를 파일로서 관리하기 위한 제어를 수행한다. 정보 저장 모듈 액세스부(538)는 FAT 처리부(532)로부터 데이터와 함께 크기와 어드레스를 건네받아 지정된 크기의 데이터를 정보 저장 모듈(550)의 저장 영역 내의 지정된 위치에 기록 하는 등, 정보 저장 모듈(550)에 대한 명령이나 데이터의 송수신을 제어한다.

[0041] FAT 처리부(532)는 또한, 통상 FAT 처리 파트(534) 및 편집 FAT 처리 파트를 포함할 수 있다. 통상 FAT 처리 파트(534)는 일반적인 방식으로 FAT를 처리하는 처리 파트이며, 편집 FAT 처리 파트(536)는 하나의 파일에서 블록 단위로 추가 및 삭제가 발생하는 경우에 FAT를 처리하는 처리 파트이다.

[0042] 정보 저장 모듈(550)은, 애플리케이션 모듈 인터페이스(552), 저장 모듈 CPU(554), 저장 모듈 RAM(556), 저장 모듈 ROM(558) 및 불휘발성 메모리(560)를 포함한다.

[0043] 애플리케이션 모듈 인터페이스(552)는, 정보 저장 모듈(550)과 통신하기 위한 인터페이스로, 제어 신호 및 데이터의 송수신을 수행한다. 저장 모듈 ROM(558)에는 정보 저장 모듈(550)을 제어하는 프로그램이 저장되어 있다.

[0044] 불휘발성 메모리(560)는 애플리케이션 모듈(510)로부터 송신된 데이터를 저장하는 영역으로, FAT 파일 시스템에 의해 관리된다. 불휘발성 메모리(560)에는 데이터 영역(572)에 저장되는 파일 및 디렉토리를 관리하기 위해, 도 1에 도시된 파일 시스템의 구조와 유사하게 마스터 부트 레코드·파티션 테이블 영역(562), 파티션 부트 영역(564), 제1 FAT 영역(566), 제2 FAT 영역(568) 및 루트 디렉토리 영역(570)이 마련되어 있다.

[0045] 도 3에 도시된 바와 같이, 애플리케이션 프로그램의 main()이 수행되면, 애플리케이션 처리부(530)는 FAT 처리부(532)의 통상 FAT 처리 파트(534)를 통해 파일 디스크립터를 생성하고, UFFS_read() 라이브러리를 동작시켜서 해당 파일을 읽어온다.

[0046] 그리고 사용자에 의해 해당 파일이 편집되면, 애플리케이션 처리부(530) 또는 FAT 처리부(532)는 편집된 파일이 데이터가 연속하여 관리되는 스트림 바이트 파일인지 아니면 블록 단위로 관리되는 파일인지를 체크한다. 그리고 FAT 처리부(532)는 편집된 파일이 데이터가 연속하여 관리되는 스트림 바이트 파일이면 통상 FAT 처리 파트(534)의 기록 라이브러리를 이용하여 정보 저장 모듈(550)에 편집된 파일을 저장하게 하고, 편집된 파일이 블록 단위로 관리되는 파일이면 편집 FAT 처리 파트(536)의 편집 라이브러리를 이용하여 정보 저장 모듈(550)에 편집된 파일을 저장하게 한다.

[0047] 만일 통상 FAT 처리 파트(534)에 의하면, 파일 오프셋 4096에 4K 바이트가 삽입되는 경우 6개의 클러스터들이 불휘발성 메모리(560)에 다시 기록될 것이다. 그러므로, 전체 24K 바이트가 불휘발성 메모리(560)에 기록될 것이고, 또한, foo 파일의 오프셋 8196에서 8K 바이트 크기 영역을 삭제한다면, 3개의 클러스터들이 불휘발성 메모리(560)에 기록될 것이다.

[0048] 하지만, 상술한 바와 같이 본 실시예에 의하면, 도 4에서 블록 단위로 추가된 경우, 먼저 사용자 레벨의 FAT 파일 시스템으로부터 프리 클러스터(클러스터 번호는 103임)를 얻고, 이어서 클러스터 오프셋으로부터 103에 4K 바이트를 복사하며, 이어서 파일 할당 테이블의 100번째 클러스터의 엔트리가 103으로 변경되고, 그리고 FAT에 추가된 블록을 링크하는 프로세서만으로 편집된 파일이 저장된다.

[0049] 또한, 도 4에서와 같이 사용자 레벨의 FAT 파일 시스템이 오프셋 8192의 foo 파일로부터 2개의 블록들을 삭제한다면, 사용자 레벨의 FAT 파일 시스템은 클러스터 번호들을 000로 변경하고 101번째 클러스터의 엔트리만을 111로 변경하면 되므로, 불휘발성 메모리(560)의 데이터 영역에 어떠한 데이터도 기록되지 않는다.

[0050] 도 5에는 ROM(520)에 애플리케이션 처리부(530), FAT 처리부(532) 및 정보 저장 모듈 액세스부(538)가 포함되어 있는 것으로 도시되어 있지만, 이들 구성은 불휘발성 메모리(560)로부터 읽어 와서 CPU(512)에 수행될 수 있다. 또한 도 5에는 정보 저장 모듈(550)에 별도의 저장 모듈 CPU(554) 등이 도시되어 있지만, 정보 저장 모듈(550)은 불휘발성 메모리(560)만으로 구성될 수 있다.

[0051] 도 6은 본 발명의 다른 실시예에 따른 파일 저장 방법의 흐름도를 도시하는 도면이다.

[0052] 정보 저장 모듈(550)은 부트 레코드 영역, 파일 할당 테이블 영역, 디렉토리 영역 및 데이터 영역을 포함하고, 파일 할당 테이블 파일 포맷에 따라 적어도 하나 이상의 파일을 저장한다(S602).

[0053] 애플리케이션 처리부(530)에서 애플리케이션 프로그램이 구동되면 FAT 처리부를 이용하여 정보 저장 모듈(550)에 저장된 해당 파일을 읽어온다(S604).

[0054] 애플리케이션 처리부(530)는 사용자의 입력에 따른 애플리케이션 프로그램에 따라 파일을 편집한다(S606). 예를 들면 Winzip 압축 파일의 경우 서브-파일들을 추가하거나 삭제할 수 있다.

[0055] 애플리케이션 처리부(530)는 편집된 파일이 블록 단위로 관리되는 파일이 아니면(S608), 즉 데이터가 연속하여 관리되는 파일이면 통상의 기록 라이브러리를 이용하여 정보 저장 모듈(550)에 편집된 파일을 저장하게 한다(S610).

[0056] 또한, 애플리케이션 처리부(530)는 편집된 파일이 블록 단위로 관리되는 파일이면 편집 라이브러리를 이용하여 정보 저장 모듈(550)에 편집된 파일을 저장하게 한다(S612). 이 경우 편집 라이브러리는 블록이 추가된 경우에는 추가된 블록의 개수만큼 정보 저장 모듈(550)의 데이터 영역 중 대응하는 개수의 클러스터들에 데이터를 저장하고 추가된 블록을 위해 파일 할당 테이블 영역의 파일 할당 테이블을 링크시키고, 블록이 삭제된 경우에는 정보 저장 모듈(550)의 데이터 영역으로의 기록 없이 삭제된 블록에 해당하는 파일 할당 테이블 영역의 파일 할당 테이블의 엔트리를 제로로 변경하고 삭제된 블록을 위해 파일 할당 테이블을 링크시킨다.

[0057] 도 7은 본 발명에 따른 파일 저장 방법의 서브-파일 관리와 통상의 파일 저장 방법의 서브-파일 관리를 비교한 그래프를 도시하는 도면이다.

[0058] 도 7에 도시된 바와 같이, 서브-파일의 관리 비교는 본 발명에 따른 bmodify() 기능을 구비한 사용자 레벨의 파일 시스템(UFFS_BM)과, bmodify() 기능을 구비하지 아니한 통상의 파일 시스템(UFFS_NBM)과, 및 리눅스 파일 시스템(EXT4)에 대해 수행되었다.

[0059] 도 7에 도시된 바와 같이, 서브-파일의 추가는 블록 기록 오버헤드 및 메타데이터 관리 오버헤드를 필요로 하므로, 처리 시간의 수행 결과는 파일 시스템들 사이에 매우 유사하였다. 파일의 중간에서의 서브-파일 삭제는 UFFS_BM이 가장 좋은 수행을 제공하였다. 하지만, 통상의 파일 시스템의 경우 EXT4 파일 시스템 상의 원래의 TAR 파일 접근과 비교하여 서브-파일 변경을 처리하기 위해 추가의 오버헤드를 야기하였다.

[0060] 본 발명의 보호 범위는 이하 특허청구범위에 의하여 해석되어야 마땅할 것이다. 또한, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자라면 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 다양한 수정 및 변형이 가능할 것인 바, 본 발명과 동등한 범위 내에 있는 모든 기술 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

부호의 설명

[0061] 510: 애플리케이션 모듈 512: CPU

514: RAM 516: 정보 저장 모듈 인터페이스

520: ROM 530: 애플리케이션 처리부

532: FAT 처리부 534: 통상 FAT 처리 파트

536: 편집 FAT 처리 파트 538: 정보 저장 모듈 액세스부

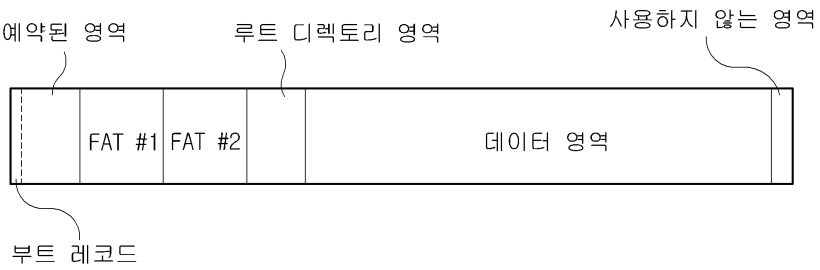
550: 정보 저장 모듈 552: 애플리케이션 모듈 인터페이스

554: 저장 모듈 CPU 556: 저장 모듈 RAM

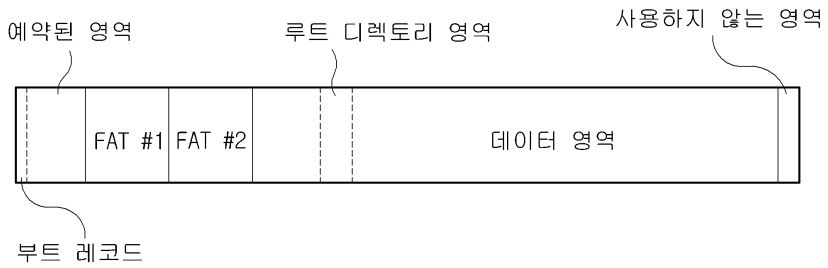
558: 저장 모듈 ROM 560: 불휘발성 메모리

도면

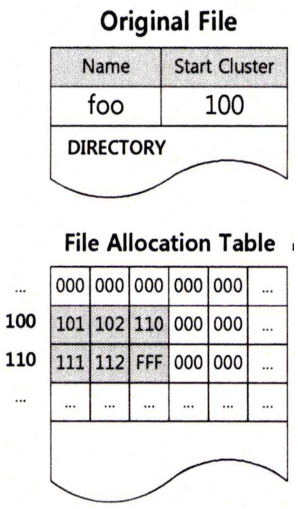
도면1a



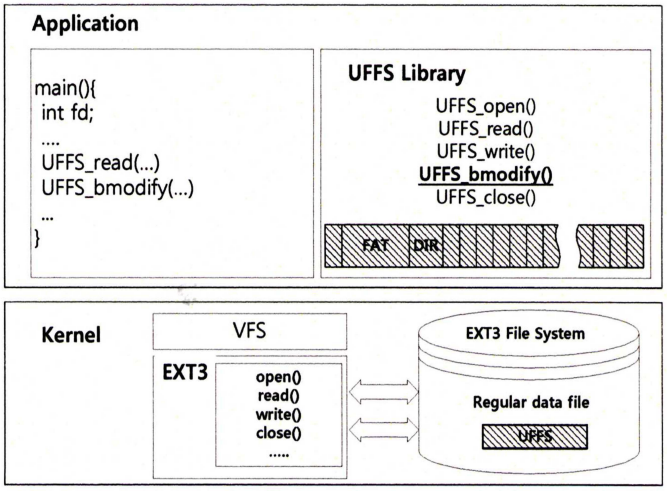
도면1b



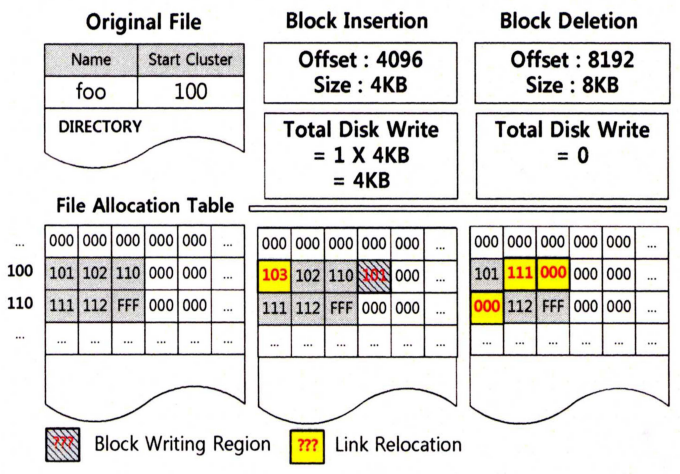
도면2


Original File
Name
Start Cluster
foo
100
DIRECTORY
File Allocation Table
...
100
110
...
000 000 000 000 000 ...
101 102 110 000 000 ...
111 112 FFF 000 000 ...
... ...

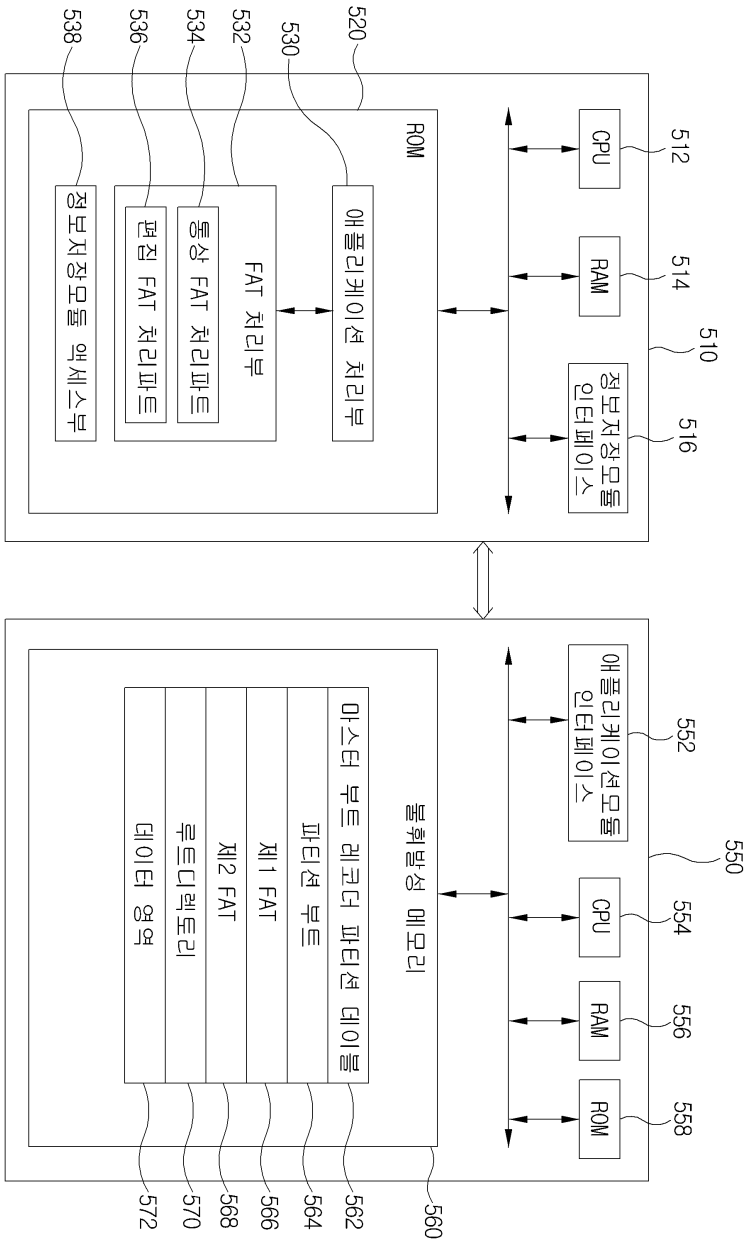
도면3


Application
main(){
int fd;
....
UFFS_read(...)
UFFS_bmodify(...)
...
}
UFFS Library
UFFS_open()
UFFS_read()
UFFS_write()
UFFS_bmodify()
UFFS_close()
FAT
DIR
Kernel
VFS
EXT3
open()
read()
write()
close()
.....
EXT3 File System
Regular data file
UFFS

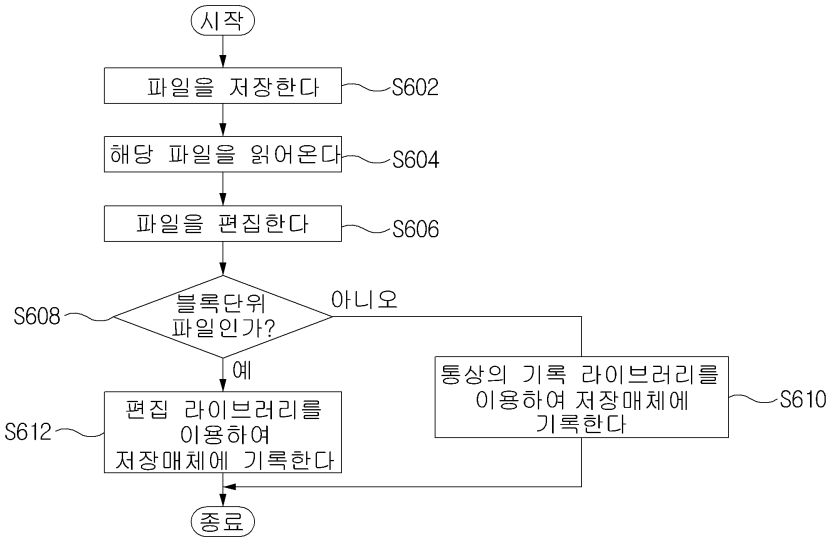
도면4


Original File
Name
Start Cluster
foo
100
DIRECTORY
Block Insertion
Offset : 4096
Size : 4KB
Total Disk Write
= 1 X 4KB
= 4KB
Block Deletion
Offset : 8192
Size : 8KB
Total Disk Write
= 0
File Allocation Table
100
110
Block Writing Region
Link Relocation

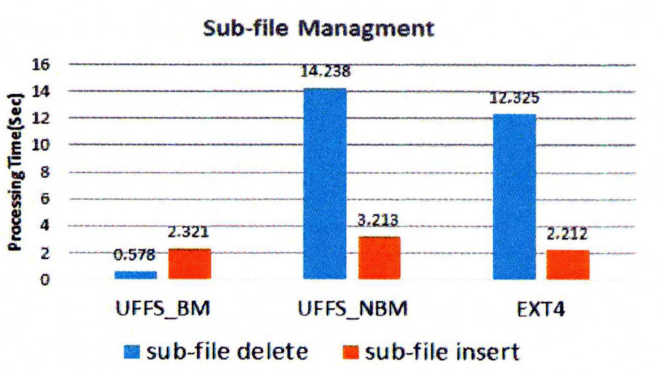
도면5


CPU
512
RAM
514
510
정보저장모듈 인터페이스
516
ROM
520
애플리케이션 처리부
530
FAT 처리부
532
통상 FAT 처리파트
534
편집 FAT 처리파트
536
정보저장모듈 액세스부
538
550
애플리케이션모듈 인터페이스
552
CPU
554
RAM
556
ROM
558
불휘발성 메모리
560
마스터 부트 레코더 파티션 데이블
562
파티션 부트
564
제1 FAT
566
제2 FAT
568
루트디렉토리
570
데이터 영역
572

도면6


시작
파일을 저장한다
S602
해당 파일을 읽어온다
S604
파일을 편집한다
S606
블록단위 파일인가?
S608
아니오
예
통상의 기록 라이브러리를 이용하여 저장매체에 기록한다
S610
편집 라이브러리를 이용하여 저장매체에 기록한다
S612
종료

도면7


Sub-file Managment
Processing Time[Sec]
16
14
12
10
8
6
4
2
0
0.578
2.321
14.238
3.213
12.325
2.212
UFFS_BM
UFFS_NBM
EXT4
sub-file delete
sub-file insert