**Chapter 15 파일 시스템**

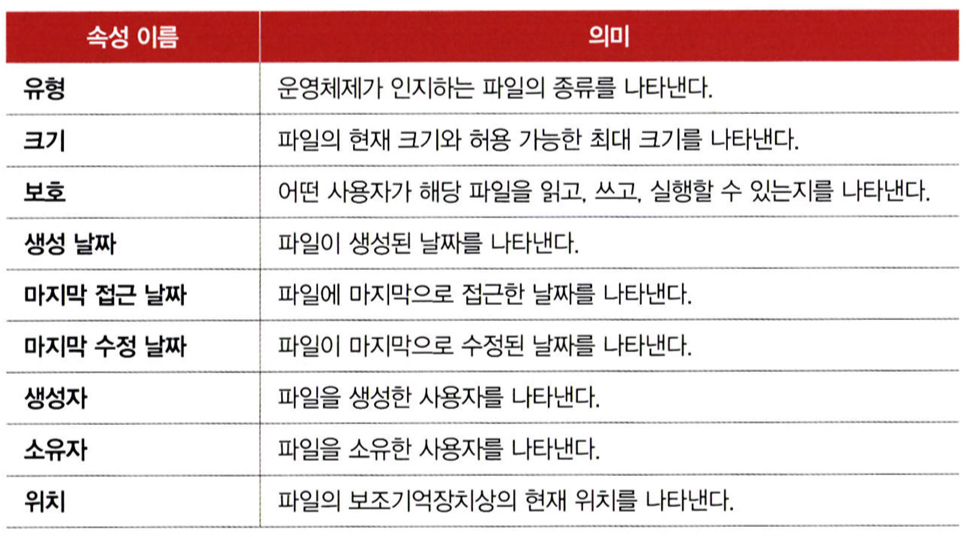
***15-1*** **파일과 디렉터리**

**<파일>**

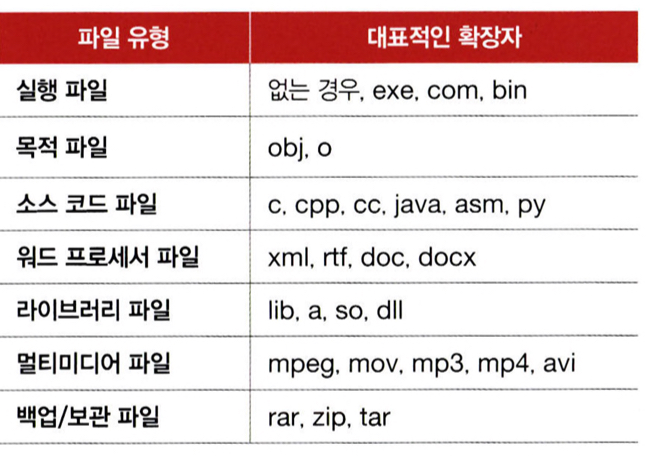
**파일 속성과 유형**

* 하드 디스크나 SSD와 같은 보조기억장치에 저장된 관련 정보의 집합
* 의미있고 관련 있는 정보를 모은 논리적 단위
* 파일을 이루는 정보에는 이름, 파일을 실행하기 위한 정보, 파일 관련 부가 정보 존재

위의 부가 정보: **속성** 또는 **메타데이터**라고 부름



* **파일 유형**: 같은 이름의 파일일지라도 텍스트 파일, 실행 파일, 음악 파일 등 유형이 다르면 실행 양상도 달라짐 -> 파일 유형을 알리기 위해 가장 흔히 사용하는 방식은 **확장자**

****

**파일 연산을 위한 시스템 호출**

파일을 다루는 모든 작업은 운영체제에 의해 이뤄짐

어떤 응용프로그램도 임의로 파일 조작 불가능, 운영체제를 통해서만 가능 -> 운영체제는 파일 연산을 위한 **시스템 호출**을 제공

* 파일 연산: 파일 생성, 삭제, 열기, 닫기, 읽기, 쓰기

**<디렉터리>**

**디렉터리**: 파일들을 일목요연 하게 관리하기 위해 사용하며, 윈도우 운영체제에서는 **폴더**라고 부름

* **1단계 디렉터리**: 옛날 운영체제에서는 하나의 디렉터리만 존재. 모든 파일이 하나의 디렉터리 아래에 있음
* **트리 구조 디렉터리**: 1단계 디렉터리로는 많은 파일을 관리하기 어렵기 때문에 생겨난 계층형 디렉터리. 최상위 디렉터리(루트 디렉터리: /)가 있고 그 아래에 여러 서브 디렉터리가 있을 수 있음.

**절대 경로와 상대 경로**

같은 디렉터리에는 동일한 이름의 파일이 존재할 수 없지만 서로 다른 디렉터리에는 동일한 이름의 파일이 존재할 수 있음 -> 루트 디렉터리부터 파일까지 경로가 다르기 때문

**절대 경로**: 루트 디렉터리에서 자기 자신까지 이르는 고유한 경로

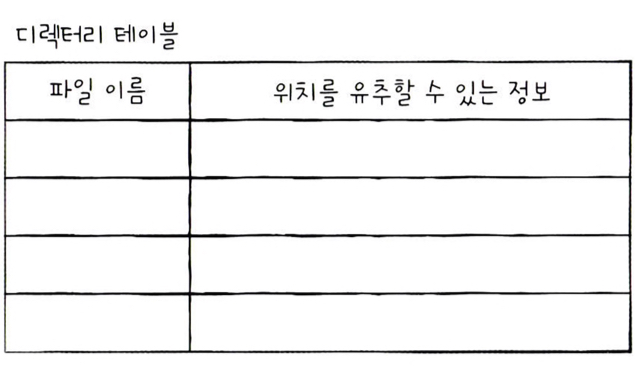
**상대 경로**: 현재 디렉터리부터 시작하는 경로

**디렉터리 연산을 위한 시스템 호출**

디렉터리는 단지 조금 특별한 정보를 포함한 파일

* 파일 -> 해당 파일과 관련된 정보를 담고 있음
* 디렉터리 -> 해당 디렉터리에 담겨 있는 대상과 관련된 정보(테이블 형태)를 담고 있음

디렉터리 엔트리가 공통으로 포함하는 정보는 디렉터리에 포함된 대상의 이름과 그 대상이 보조기억장치 내에 저장된 위치를 유추할 수 있는 정보



***15-2*** **파일 시스템**

파일 시스템에는 다양한 종류가 있고, 하나의 컴퓨터에서 여러 파일 시스템을 사용 가능

대표적인 파일 시스템은 **FAT 파일 시스템**, **유닉스 파일 시스템**이 있음

**<파티셔팅과 포매팅>**

새 하드 디스크 또는 SSD의 보조기억장치에 곧바로 파일 생성 및 저장이 불가능

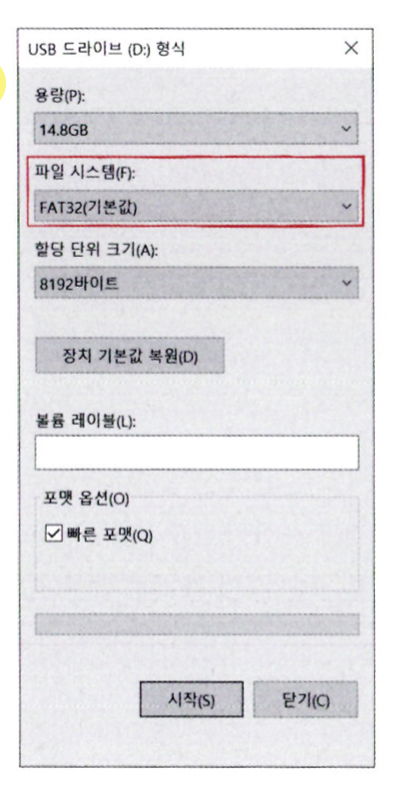
-> 보조기억장치를 사용하려면 파티션을 나누는 작업**(파티셔닝**)과 포맷작업(**포매팅**)을 거쳐야 함

**파티셔닝**: 저장 장치의 논리적인 영역을 구획하는 작업 (칸막이로 영역을 나누는 작업)

* 파티셔닝 작업을 통해 나누어진 영역 하나하나를 **파티션**이라고 함

**포매팅**: 파일 시스템을 설정하여 어떤 방식으로 파일을 저장하고 관리할 것인지를 결정하고 새로운 데이터를 쓸 준비를 하는 작업

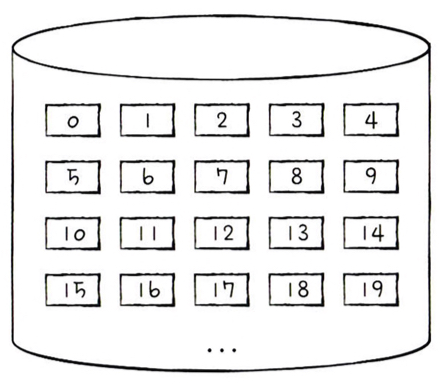
* **저수준 포매팅**: 저장 장치를 생성할 당시 공장에서 수행되는 물리적인 포매팅
* **논리적 포매팅**: 파일 시스템을 생성하는 포매팅



USB 메모리 포매팅

**<파일 할당 방법>**

* 운영체제는 파일과 디렉터리를 **블록** 단위로 읽고 씀
* 하나의 파일이 보조기억장치에 저장될 때는 하나 이상의 블록에 걸쳐 저장
* 하드 디스크의 가장 작은 저장 단위는 섹터이지만 운영체제는 블록 단위로 파일과 디렉터리를 관리 (섹터 단위로 관리하기에는 개수가 너무 많고 크기도 작기 때문)



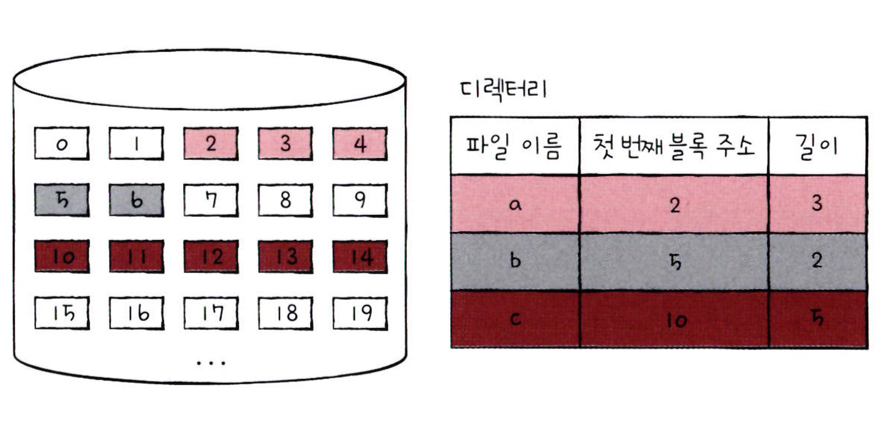
* 블록 안에 적힌 번호: 주소
* 크기가 작은 파일은 적은 수의 블록에 걸쳐 저장, 크기가 큰 파일은 여러 블록에 걸쳐 저장

파일을 보조기억장치에 할당하는 방법은 두 가지 -> **연속 할당**, **불연속 할당**

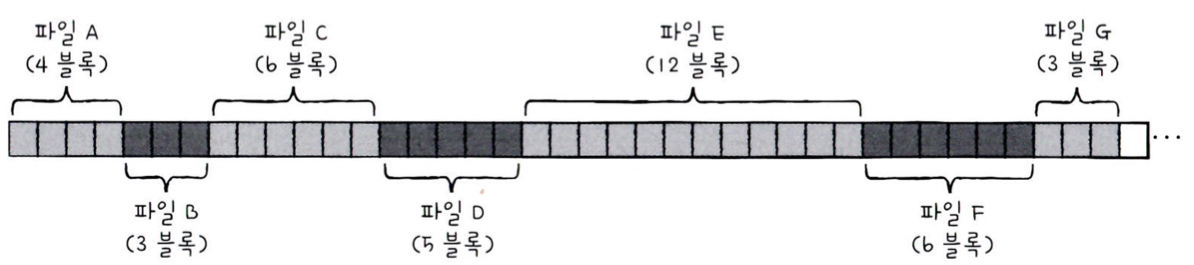
불연속 할당 방법 두 가지 -> **연결 할당**, **색인 할당**

**연속 할당**

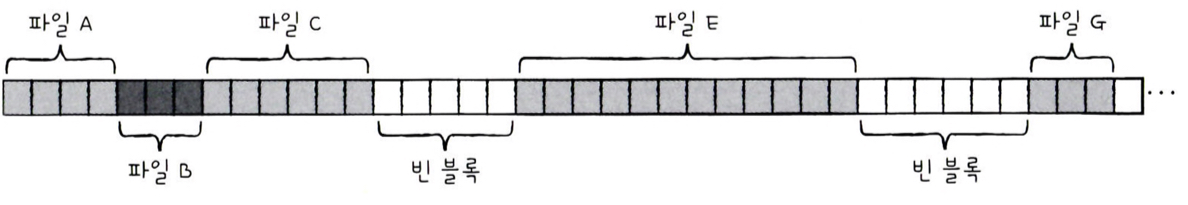
* 가장 단순한 방식
* 보조기억장치 내 연속적인 블록에 파일을 할당하는 방식
* 연속으로 할당된 파일에 접근하기 위한 정보: 첫 번째 블록 주소, 블록 단위의 길이



* **장점**:구현이 단순, **단점** :외부 단편화를 야기



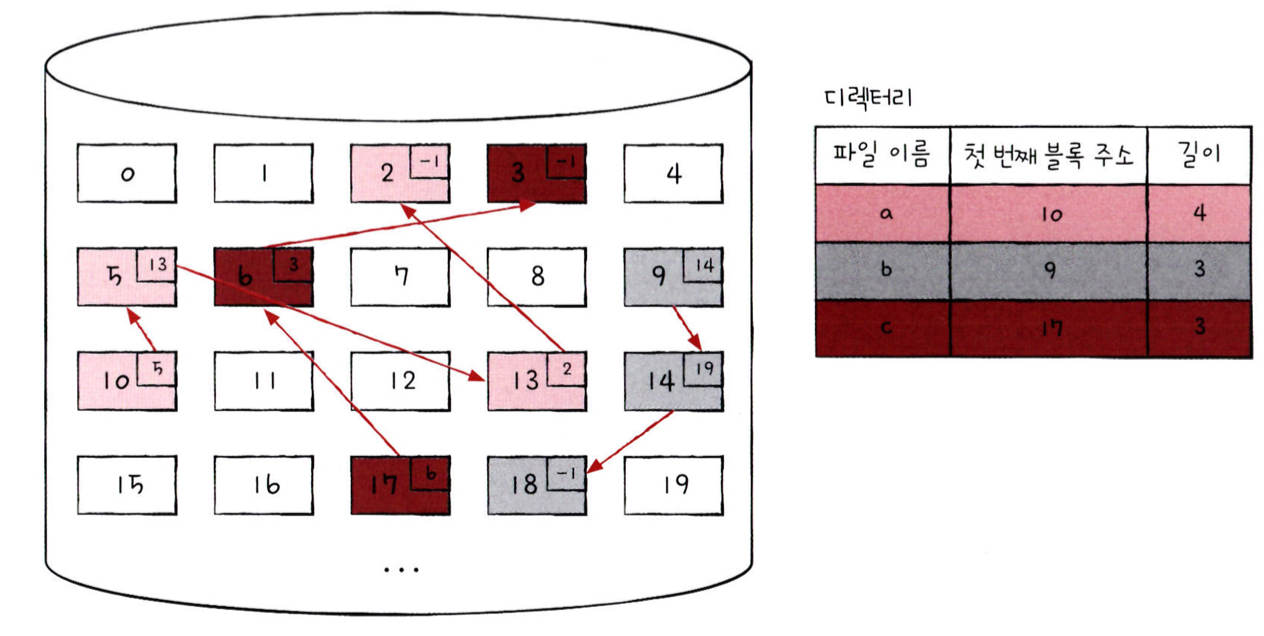
연속 할당 시



파일 D,F 삭제 시

**연결 할당**

* 연속 할당의 문제를 해결할 수 있는 방식(불연속 할당의 일종)
* 각 블록 일부에 다음 블록의 주소를 저장하여 각 블록이 다음 블록을 가리키는 형태로 할당하는 방식
* 디렉터리 엔트리에 명시되는 정보 -> 첫 번째 블록 주소, 블록 단위의 길이



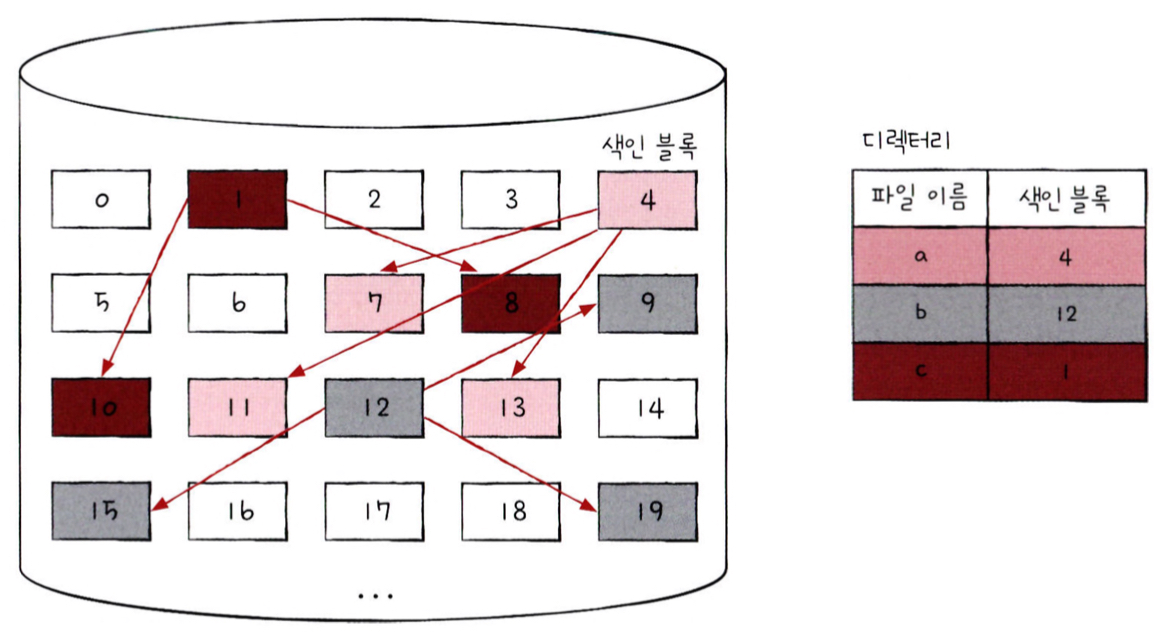
* 외부 단편화 문제를 해결하지만 단점이 있음

1. 반드시 첫 번째 블록부터 하나씩 차례대로 읽어야 함 -> 파일 내 임의의 위치에 접근하는 속도(임의 접근 속도)가 매우 느림, 성능 면에서 상당히 비효율적
2. 하드웨어 고장이나 오류 발생 시 해당 블록 이후 블록은 접근 불가

* 연결 할당을 변형한 대표적인 파일시스템이 **FAT 파일 시스템**

**색인 할당**

* 파일의 모든 블록 주소를 색인 블록이라는 하나의 블록에 모아 관리하는 방식
* 파일 내 임의의 위치에 접근하기 쉬움
* 디렉터리 엔트리에 명시되는 정보: 파일 이름, 색인 블록 주소

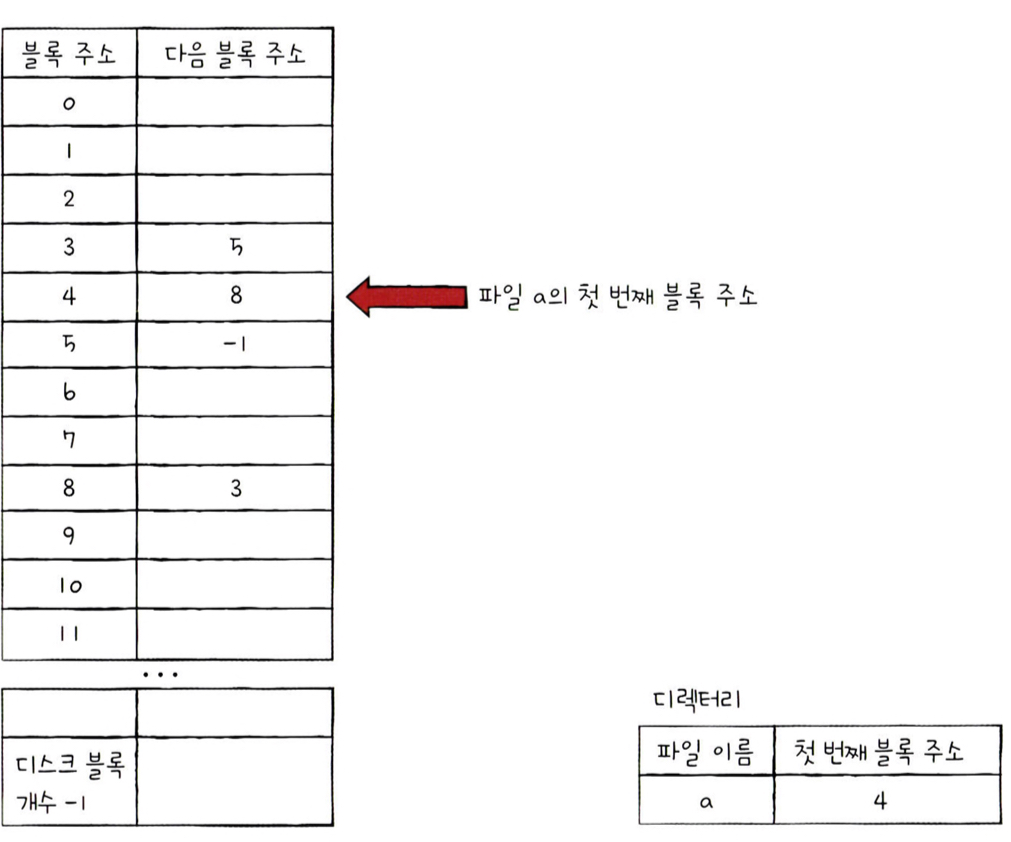


* 색인 할당을 기반으로 만든 파일 시스템이 **유닉스 파일 시스템**

**<파일 시스템 살펴보기>**

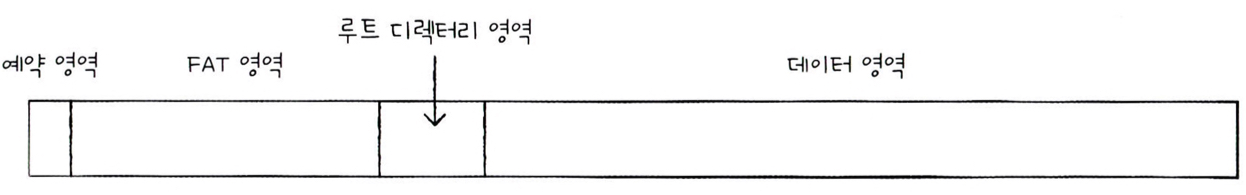
**FAT 파일 시스템**

* USB 메모리, SD 카드 등의 저용량 저장 장치에서 사용
* 연결 할당의 단점은 보완한 파일 시스템
* 각 블록에 포함된 다음 블록의 주소들을 한데 모아 테이블 형태로 관리(파일 할당 테이블 FAT)

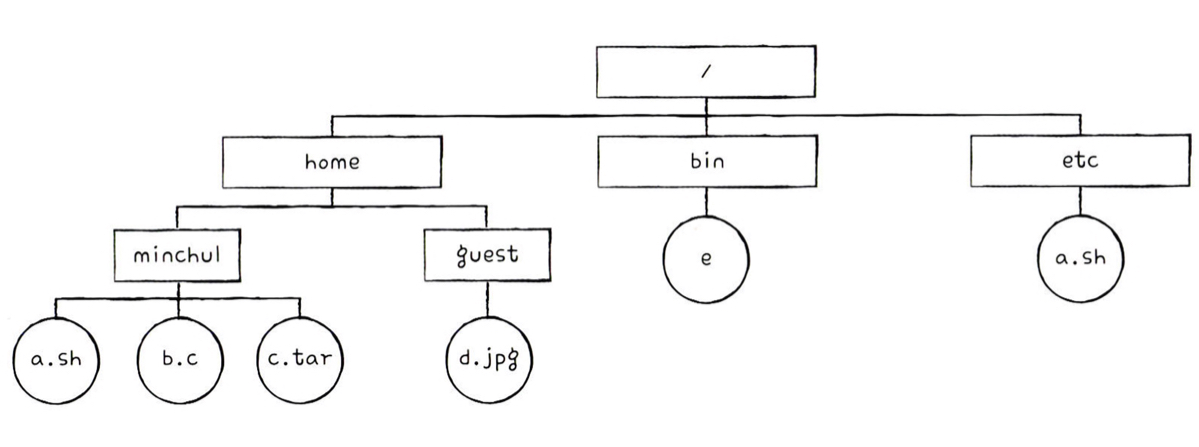


* 버전에 따라 FAT12, FAT16, FAT32가 있으며 뒤에 오는 숫자는 블록을 표현하는 비트 수
* FAT는 파티션의 앞부분에 만들어짐

(FAT 영역 - 루트 디렉터리 영역 - 서브 디렉터리와 파일들을 위한 영역)



* 실행하는 도중 FAT가 메모리에 캐시될 수 있음 -> 임의 접근의 성능이 개선



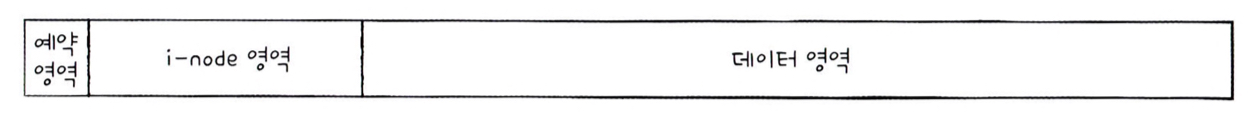
/home/minchul/a.sh 에 접근할 경우



**유닉스 파일 시스템**

* 색인 할당 기반
* 유닉스 파일 시스템에서는 색인 블록을 i-node라고 함
* i-node에는 파일 속성 정보와 열다섯 개의 블록 주소가 저장될 수 있음
* 파일마다 i-node가 있고 i-node마다 번호가 부여되어 파티션 내 특정 영역에 모여 있음

(i-node 영역 - 데이터 영역)



* 문제점: i-node의 크기는 유한함. i-node 하나는 기본적으로 열 다섯개의 블록 주소 저장 가능 -> 블록 15개 이상을 차지하는 크기가 큰 파일은 i-node 하나만으로 데이터 블록을 모두 가리킬 수 없음
* 해결 방법

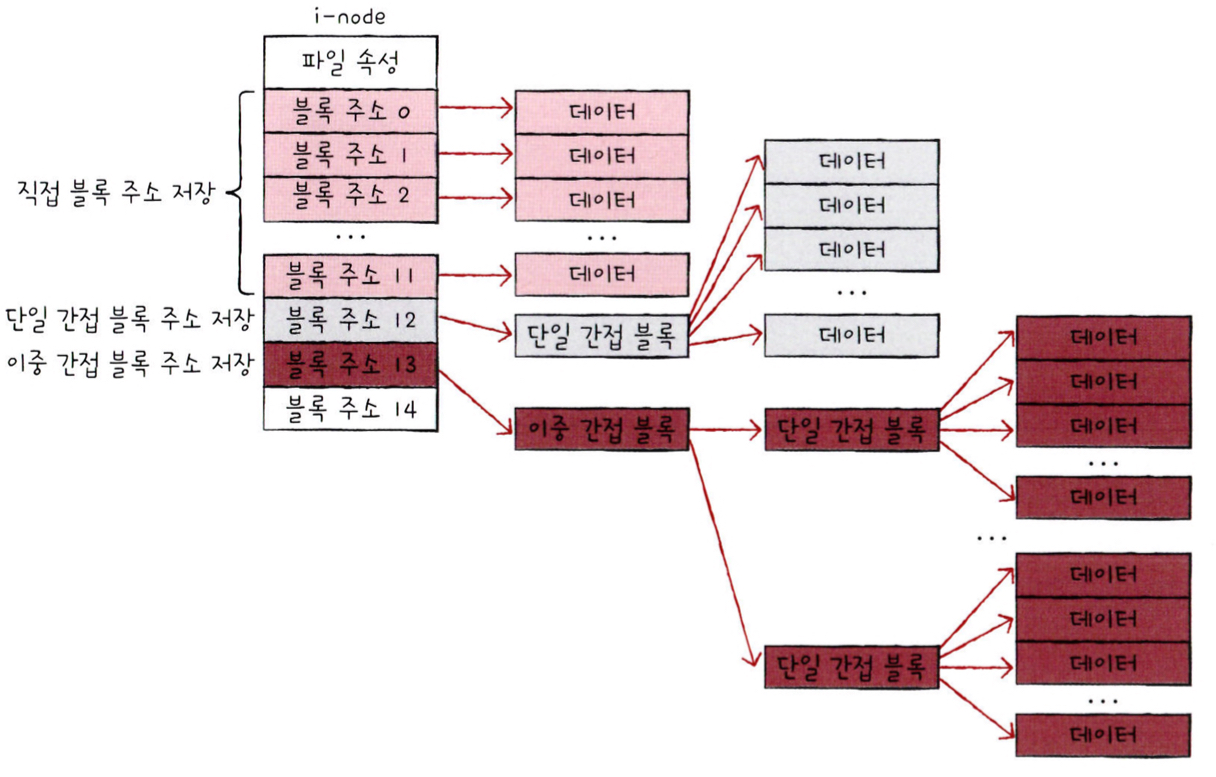
1. 블록 주소 중 열두 개에는 직접 블록 주소를 저장(직접 블록)
2. 열두 개로 충분하지 않은 경우 열세번째 주소에 단일 간접 블록 주 소를 저장

* **단일 간접 블록**: 파일 데이터가 저장된 블록이 아닌 파일 데이터를 저장한 블록 주소가 저장된 블록



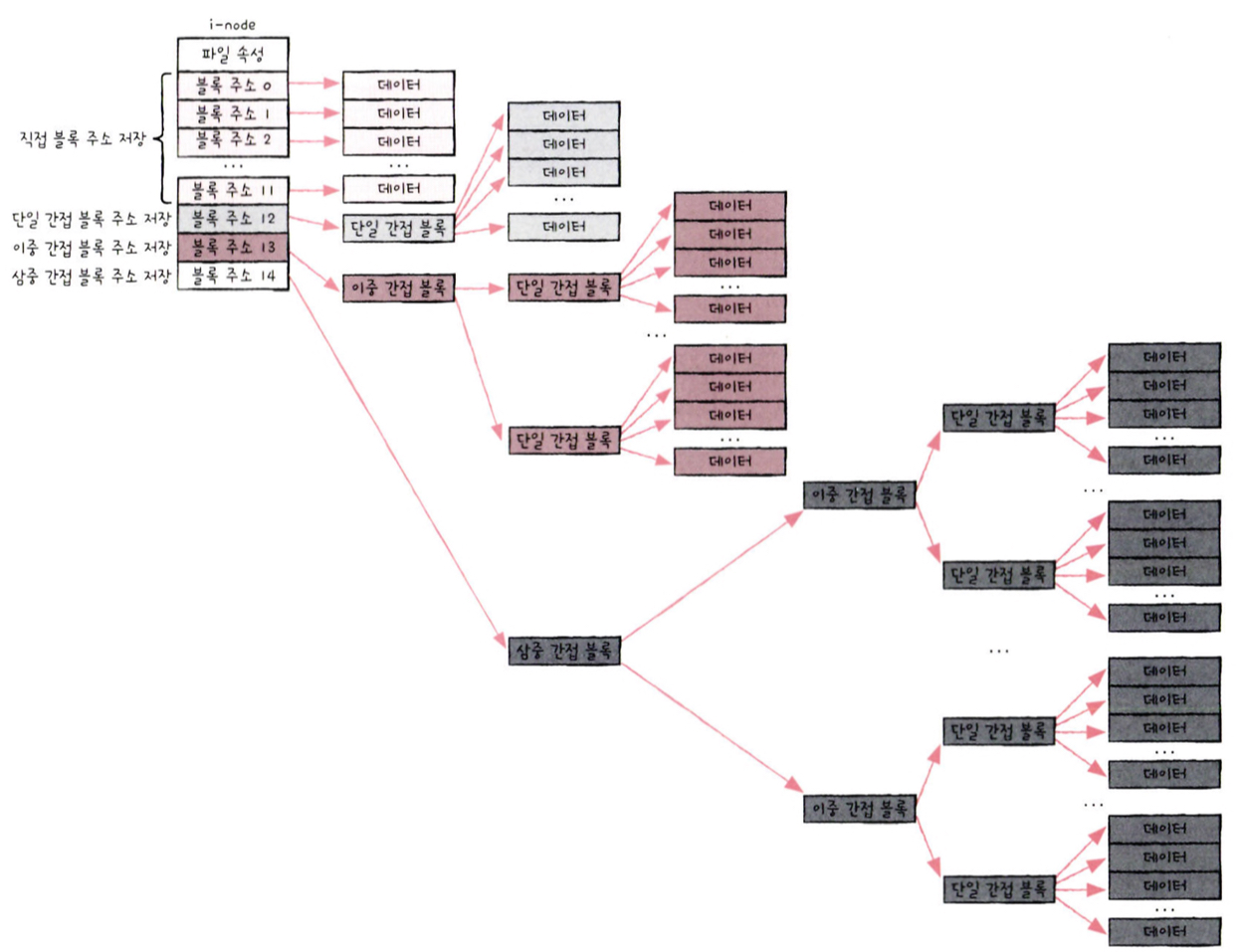
1. 단일 간접 블록으로도 충분하지 않다면 열네 번째 주소에 이중 간접 블록 주소를 저장

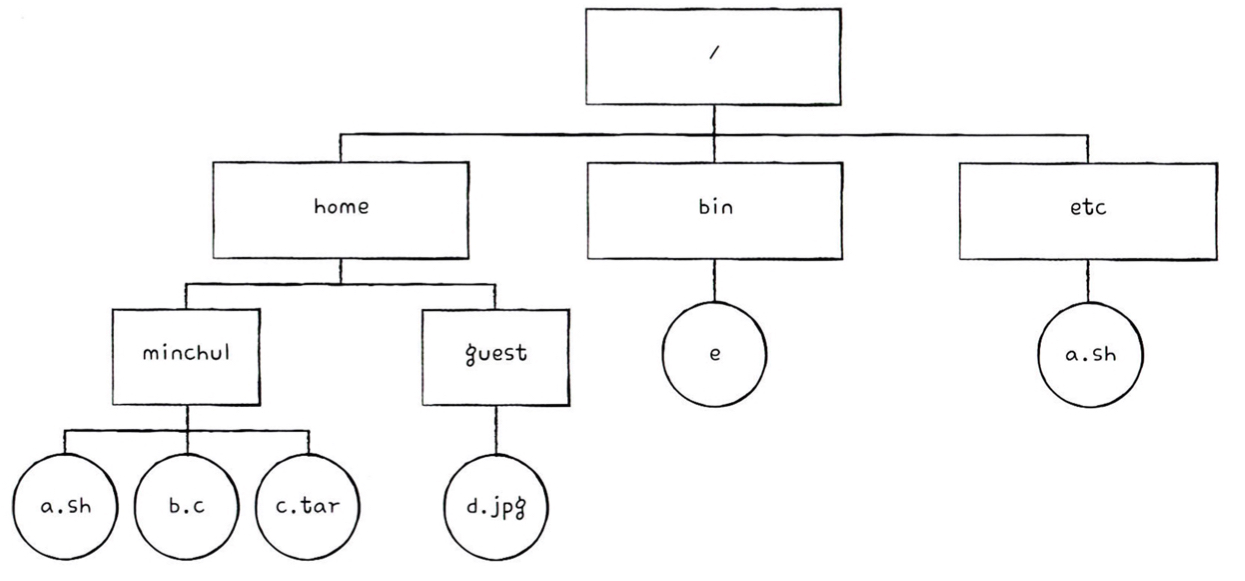
* **이중 간접 블록**: 데이터 블록 주소를 저장하는 블록의 주소가 저장된 블록



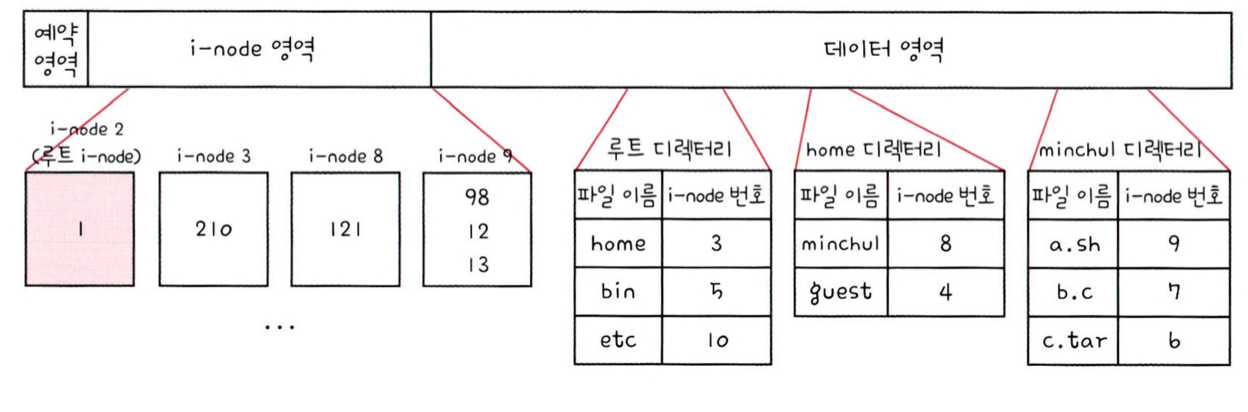
1. 이중 간접 블록으로도 충분하지 않다면 열다섯 번째 주소에 삼중 간접 블록 주소를 저장

* 삼중 간접 블록: 이중 간접 블록의 주소가 저장된 블록





/home/minchul/a.sh 에 접근할 경우



**<저널링 파일 시스템>**

컴퓨터가 강제 종료 되었을 때 파일 시스템을 변경하는 중이라면 파일 시스템이 훼손될 수 있음

저널링 파일 시스템이 있기 전: 부팅 직후 파일 시스템을 검사하고 복구하는 프로그램 실행

-> 시간이 너무 오래 걸림

**저널링 파일시스템**: 작업 로그를 남겨 시스템 크래시 상황에서 빠르게 복구 가능

-> 시스템 크래시가 발생한 직후에 로그 영역을 읽어 크래시가 발생한 당시 어떤 작업을 실행 중이었는지 알아낸 다음 해당 작업을 완료

**<마운트>**

한 저장 장치의 파일 시스템에서 다른 저장 장치의 파일 시스템에 접근할 수 있도록 파일 시스템을 편입시키는 작업