



15장 - 파일 시스템

◆ 15-1. 파일과 디렉터리

- 파일과 디렉터리는 모두 운영체제 내부 파일 시스템이 관리하는 존재
- 파일과 디렉터리는 보조기억장치에 있는 데이터 덩어리일뿐인데, 운영체제는 이를 어떻게 파일과 디렉터리로 관리할까?
- 파일 시스템이 관리하는 파일과 디렉터리에 대해 학습

◎ 파일

- 하드 디스크나 SSD와 같은 보조기억장치에 저장된 관련 정보의 집합을 의미
- 의미 있고 관련 있는 정보를 모은 논리적 단위

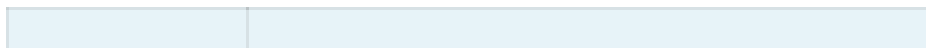
◎ 파일을 이루는 정보

- 모든 파일에는 이름과 파일을 실행하기 위한 정보, 그리고 파일 관련 부가 정보 ⇒ 속성 또는 메타데이터



- 파일형식, 위치, 크기 등 파일과 관련된 다양한 정보가 나타남 → 파일 속성

◎ 파일 속성



속성 이름	의미
유형	운영체제가 인지하는 파일의 종류
크기	파일의 현재 크기와 허용 가능한 최대 크기를 나타냄
보호	어떤 사용자가 해당 파일을 읽고, 쓰고, 실행할 수 있는지를 나타낸다.
생성 날짜	파일이 생성된 날짜를 나타냄
마지막 접근 날짜	파일에 마지막으로 접근한 날짜를 나타냄
마지막 수정 날짜	파일이 마지막으로 수정된 날짜
생성자	파일을 생성한 사용자
소유자	파일을 소유한 사용자
위치	파일의 보조기억장치상의 현재 위치

◎ 파일 유형

- 파일 속성 중 파일 유형은 운영체제가 인식하는 파일 종류
- 같은 이름의 파일이더라도 텍스트 파일, 실행 파일, 음악 파일 등 다를 수 있다
 - joon.exe
 - joon.py
 - joon.mp4
- 파일 유형을 알리기 위해 가장 흔히 사용되는 방식 → **확장자 (extension)**

파일 유형	대표적인 확장자
실행 파일	없는 경우, exe, com, bin
목적 파일	obj, o
소스 코드 파일	c, cpp, cc, java, asm, py
워드 프로세서 파일	xml, rtf, doc, docx
라이브러리 파일	lib, a, so, dll
멀티미디어 파일	mpeg, mov, mp3, mp4, avi
백업/보관 파일	rar, zip, tar

- 위에서 몇몇 확장자만 알아보자

▼ xml

- eXtensible Markup Language (다목적 마크업 언어)
- W3C에서 개발된 다른 특수한 목적을 갖는 마크업 언어
- XML은 주로 다른 종류의 시스템, 특히 인터넷에 연결된 시스템끼리 데이터를 쉽게 주고 받을 수 있게 하여 HTML의 한계를 극복할 목적으로 만들어졌다.

▼ rtf

- 서식 있는 텍스트 포맷, 리치 텍스트 포맷
- 마이크로소프트사가 1987년에 개발한 규격인 사유의 문서 파일 형식이며 크로스 플랫폼 문서 교환을 위하여 만들어졌다.
- 대부분의 문서 처리 프로그램은 일부 버전의 RTF를 읽고 쓸 수 있음

▼ dll

- 동적 링크 라이브러리 (dynamic-link library)
- 마이크로소프트 윈도우에서 구현된 동적 라이브러리
- 내부에는 다른 프로그램이 불러서 쓸 수 있는 다양한 함수들을 가지고 있는데, 확장 DLL인 경우는 클래스를 가지고 있기도 한다. DLL은 COM(컴퍼넌트 오브젝트 모델 - active X, OLE 등의 기술을 포함하는 포괄적 개념)을 담는 그릇의 역할도 한다.

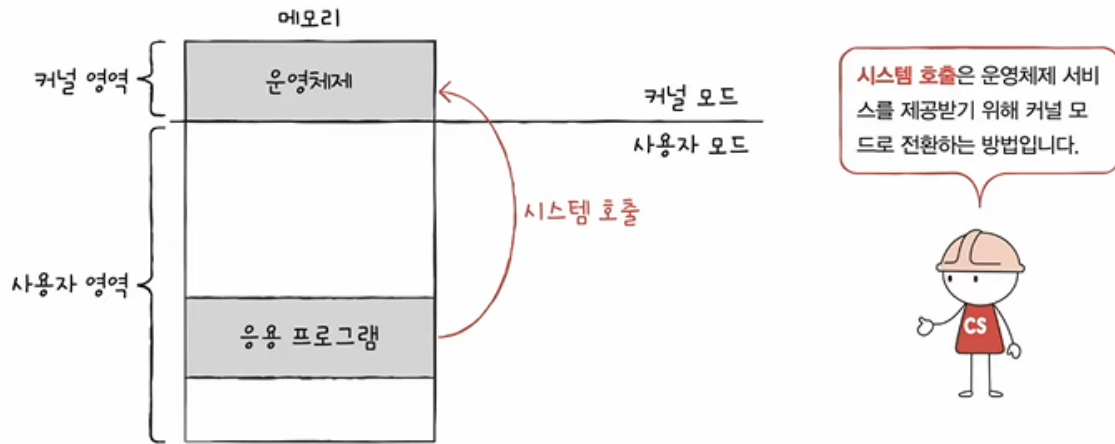
◎ 파일 연산을 위한 시스템 호출

- 어떤 응용 프로그램도 임의로 파일을 조작할 수 없으며, 파일을 다루려면 운영체제에 부탁해야 합니다.
- 이를 위해 운영 체제는 **파일 연산(생성, 삭제, 열기, 닫기, 읽기, 쓰기)**을 위한 **시스템 호출**을 제공

▼ 시스템 호출이 무엇인지?

◎ 시스템 호출

- 사용자 모드로 실행되는 프로그램이 자원에 접근하는 운영체제 서비스를 제공 받으려면 운영체제에 **요청**을 보내 커널모드로 전환되어야 함 → 이 요청이 **시스템 호출**
- 커널 모드로 전환하여 실행하기 위해 호출
- 일종의 소프트웨어 인터럽트



◎ 디렉터리

- 파일들을 일목요연하게 관리하기 위해 **디렉터리**를 이용한다
- 윈도우 운영체제의 디렉터리 ⇒ **폴더**
- 옛날 운영체제는 하나의 디렉터리만 존재
- 모든 파일이 하나의 디렉터리 아래 있었음 ⇒ 1단계 디렉터리
- 하나의 디렉터리로는 많은 파일 관리 어렵기 때문에 여러 계층을 가진 **트리 구조 디렉터리**가 등장하게 됨
- 트리구조 디렉터리
- **최상위 디렉터리 + 여러 서브 디렉터리 (자식 디렉터리)**
- 서브 디렉터리도 또 다른 서브 디렉터리를 가질 수 있음
- 최상위 디렉터리 → **루트 디렉터리** 라고 부르고, **슬래시 (/)** 로 표현함

◎ 절대 경로와 상대 경로

- 같은 디렉터리에는 동일한 이름의 파일이 존재할 수 없지만,
- 서로 다른 디렉터리에는 동일한 이름의 파일이 존재할 수 있다.
- **절대 경로** : 모든 파일은 루트 디렉터리에서 자기 자신까지 이르는 고유한 경로를 가짐

- 유닉스, 리눅스, mac OS 등의 운영체제에서는 슬래시 기호(/)는 루트 디렉터리 표시 및 디렉터리와 디렉터리 사이의 구분자로도 사용
- 윈도우에서는 디렉터리 구분자로 \ 사용 (원화 표시)
- 상대 경로 → 그림에서 현재 디렉터리 경로가 /home이라면 d.jpg의 파일 상대 경로는 guest/d.jpg 가 된다.

◎ 디렉터리 연산을 위한 시스템 호출

- 운영체제는 디렉터리 연산을 위한 시스템 호출도 제공
- 디렉터리 생성, 삭제, 열기, 닫기, 읽기

◎ 디렉터리 엔트리

- 많은 운영체제에서 디렉터리를 그저 '특별한 형태의 파일'로 간주한다.
- 즉, 디렉터리도 파일이다. 단지 포함된 정보가 조금 특별할 뿐
- **파일** ← 내부에 해당 파일과 관련된 정보를 담음
- **디렉터리** ← 내부에 해당 디렉터리에 담겨 있는 대상과 관련된 정보를 담음
- 디렉터리의 이 정보는 보통 테이블 (표) 형태로 구성
- **디렉터리는 보조기억장치에 테이블 형태의 정보로 저장됨**
- 각각의 엔트리 (행)에 담기는 정보는 파일 시스템마다 차이가 있음
- 파일 시스템 별 디렉터리 엔트리 → 다음 절에서 학습

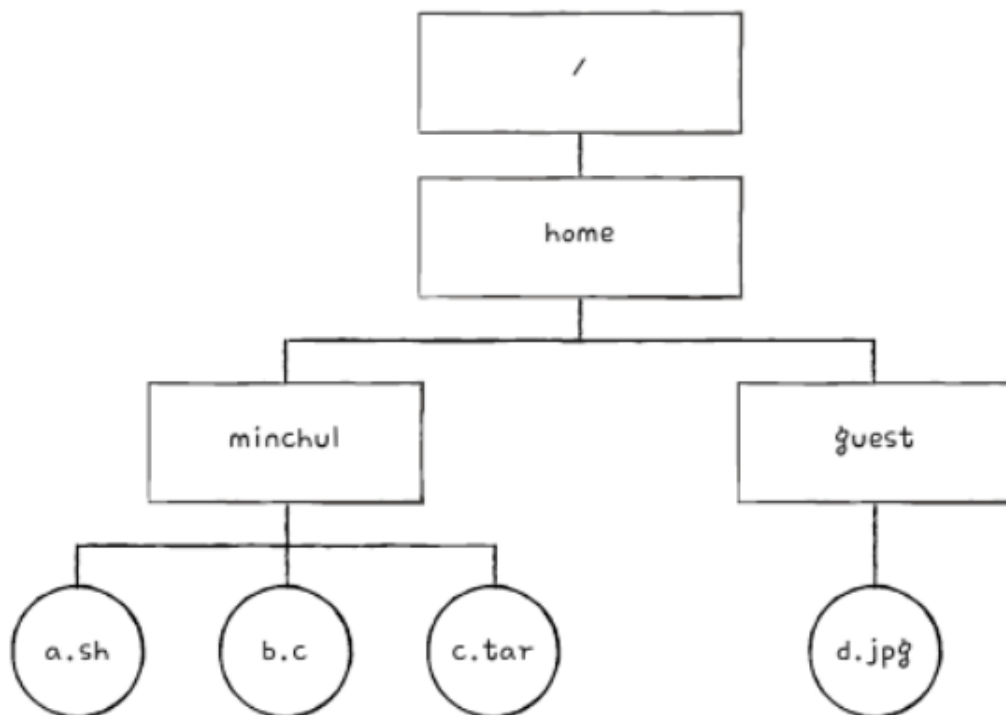
디렉터리 테이블

파일 이름	위치를 유추할 수 있는 정보

디렉터리 테이블

파일 이름	위치를 유추할 수 있는 정보	생성 시간	수정된 시간	크기	...

- 보조기억 장치 내의 파일 및 디렉터리 위치를 나타내는 방법에는 파일 시스템마다 차이가 있음

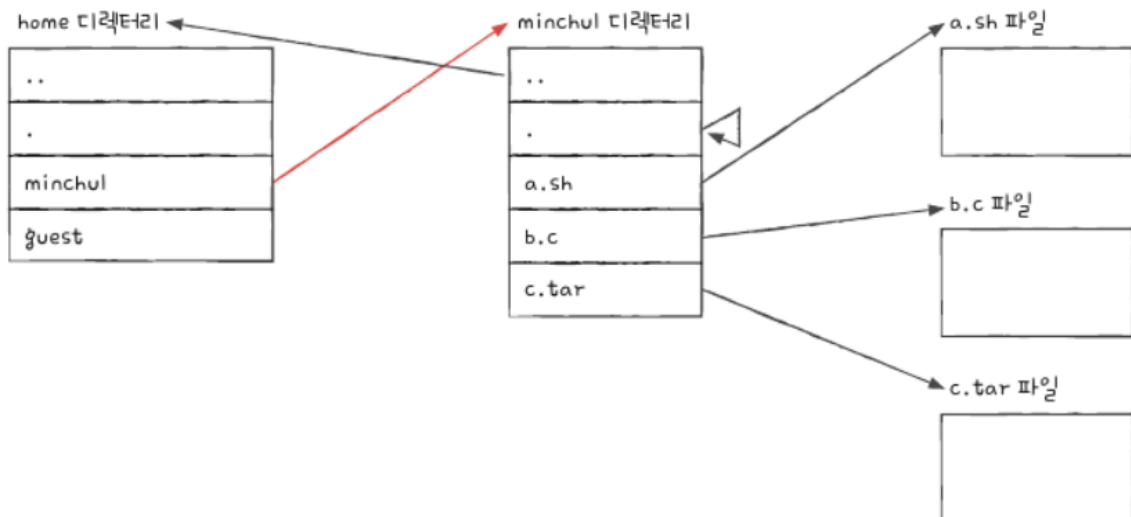


- .. 은 상위 디렉터리, .은 현재 디렉터리

home 디렉터리 테이블

파일 이름	위치를 유추할 수 있는 정보
..	
.	
minchul	
guest	

- 디렉터리 엔트리를 통해 보조기억장치에 저장된 위치를 파악 가능하고 실행할 수 있음



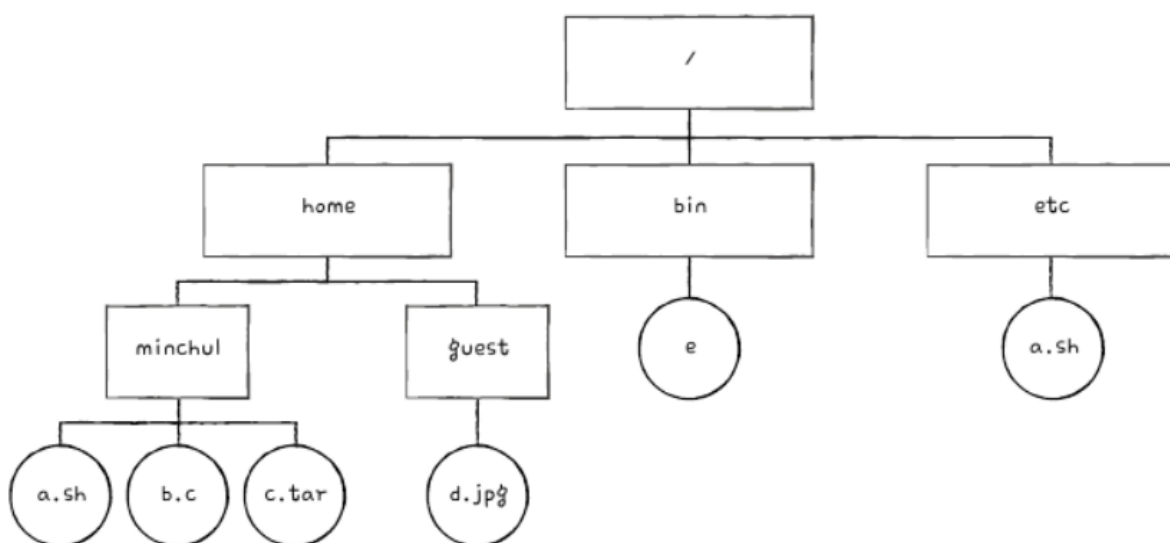
◎ 상대 경로를 나타내는 또다른 방법

- 대부분의 운영체제는 **현재 작업 디렉터리**를 마침표 (.), 현재 작업 디렉터리의 상위 디렉터리(**부모 디렉터리**)를 마침표 두 번(..)으로 나타냄
- `cd ..` 을 입력하면 부모 디렉터리로 이동함


```
명령 프롬프트
C:\W>cd C:\Wtest
C:\Wtest>cd ..
C:\W>
```

- 다만 윈도우 루트 디렉터리(C:\W)는 부모 디렉터리가 없기 때문에 cd ..을 입력해도 현재 작업 디렉터리는 변하지 않음
- ▼ 그럼 D:\W 같은 경우는?

◎ 깜짝 문제



1. 현재 작업 디렉터리가 /home일 때, c.tar의 상대 경로는?

▼ 정답

- minchul/c.tar

2. 현재 작업 디렉터리가 /home일 때, c.tar의 절대 경로는?

▼ 정답

- /home/minchul/c.tar

- 절대 경로는 루트 디렉터리부터 시작하는 경로
- 상대 경로는 현재 디렉터리부터 시작하는 경로

◆ 15-2. 파일 시스템

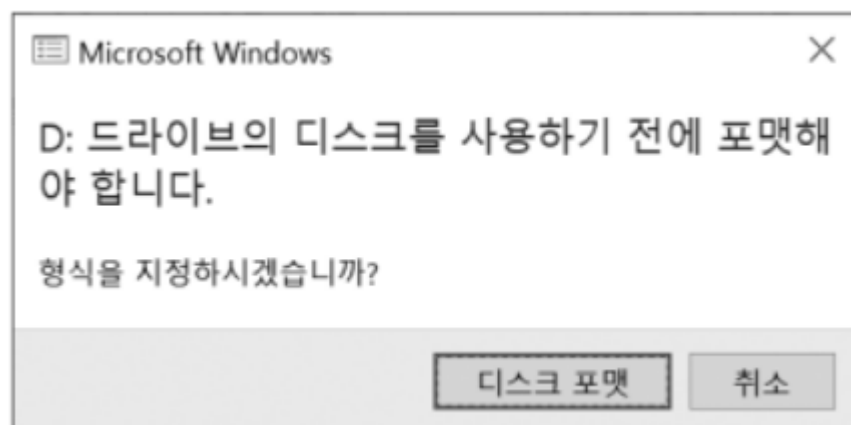
- **파일 시스템** : 파일과 디렉터리를 보조기억장치에 일목요연하게 저장하고 접근할 수 있게하는 운영체제 내부 프로그램
- 파일 시스템이 파일과 디렉터리를 보조기억장치에 어떻게 할당하고 접근하는 지에 관한 이론적인 내용 먼저 학습
- 이러한 이론을 기반으로 만들어진 대표 파일 시스템 → **FAT 파일 시스템, 유닉스 파일 시스템** 학습

◎ 파티셔닝과 포매팅

- 공장에서 이제 막 생산되어 한번도 사용된적 없는 새 하드 디스크 혹은 SSD

→ 이 보조기억장치에 곧바로 파일을 생성하거나 저장할 수 없음

- 보조기억장치를 사용하려면 **파티션**을 나누는 작업(**파티셔닝**)과 **포맷** 작업(**포매팅**)을 해야하기 때문



◎ 파티셔닝

- partitioning

partition (partitioning)

미국식 [pa:r | tɪʃn]  영국식 [pa: | tɪʃn] 


명사

1 칸막이

a glass **partition** 

유리 칸막이


2 (한 국가의) 분할

the **partition** of Germany after the war 

전후 독일의 분할

동사

1 분할하다, 나누다

to **partition** a country 

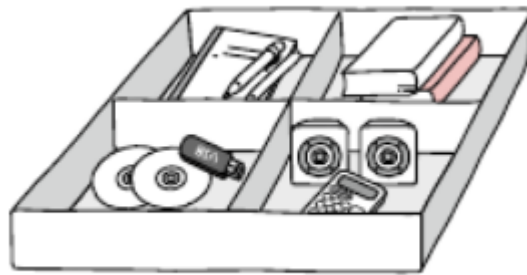
국가를 분할하다

영어사전 다른 뜻 1

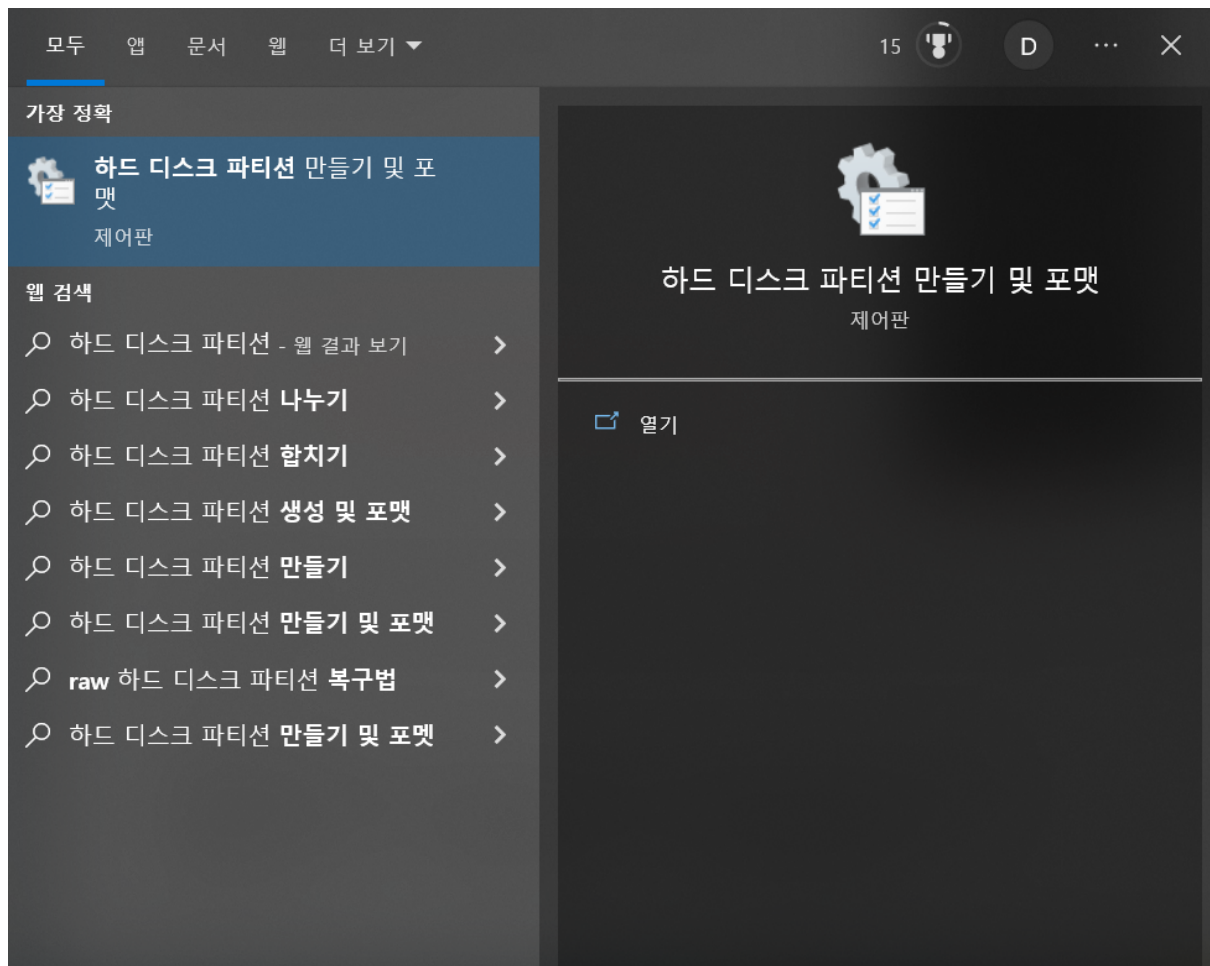
- 저장 장치의 논리적인 영역을 구획하는 작업을 의미



- 이런 서랍 안에 칸막이를 설치하여 영역을 나누면 물건들을 정리하기가 조금 더 수월해짐



- 칸막이를 영역으로 나누는 작업 → 파티셔닝
- 파티셔닝작업을 통해 나누어진 영역 하나하나 → 파티션



- 이곳에 들어가면 컴퓨터 내의 보조기억장치가 어떻게 파티셔닝 되어 운영되는지 한 눈에 볼 수 있음

디스크 관리							
파일(F) 동작(A) 보기(V) 도움말(H)							
볼륨	레이아웃	형식	파일 시스템	상태	용량	사용 가...	사용 가능한...
(디스크 0 파티션 1)	단순	기본		정상 (EFI ...	260 MB	260 MB	100 %
(디스크 0 파티션 4)	단순	기본		정상 (복구...	850 MB	850 MB	100 %
(디스크 0 파티션 5)	단순	기본		정상 (복구...	15.25 GB	15.25 GB	100 %
(디스크 0 파티션 6)	단순	기본		정상 (복구...	1.00 GB	1.00 GB	100 %
Windows10 (C:)	단순	기본	NTFS	정상 (부팅...	936.52 GB	861.50 ...	92 %

디스크 0 기본 953.85 GB 온라인	260 MB 정상 (EFI 시...	Windows10 (C:) 936.52 GB NTFS 정상 (부팅, 페이지 파일, 크래시 덤...)	850 MB 정상 (복구 파티...	15.25 GB 정상 (복구 파티션)	1.00 GB 정상 (복구 파티션)
--	------------------------	--	------------------------	-------------------------	------------------------

◎ 포매팅

- 포매팅을 저장 장치를 완전히 삭제하는 것으로 알고 있는 사람들이 많음 → 완벽하게 정확한 표현이라고 보기 어려움
- **포매팅** : 파일 시스템을 설정하여 어떤 방식으로 파일을 저장하고 관리할 것인지를 결정하고, 새로운 데이터를 쓸 준비를 하는 작업을 의미
- 어떤 종류의 파일 시스템을 사용할지 이때 결정됨
- 포매팅의 종류
 - **저수준 포매팅** : 저장 장치를 생성할 당시 공장에서 수행되는 물리적 포매팅
 - **논리적 포매팅** : 파일 시스템을 생성하는 포매팅 (여기서 설명하는건 이것)

USB 드라이브 (D:) 형식

용량(P):
14.8GB

파일 시스템(F):
FAT32(기본값)

할당 단위 크기(A):
8192바이트

장치 기본값 복원(D)

볼륨 레이블(L):

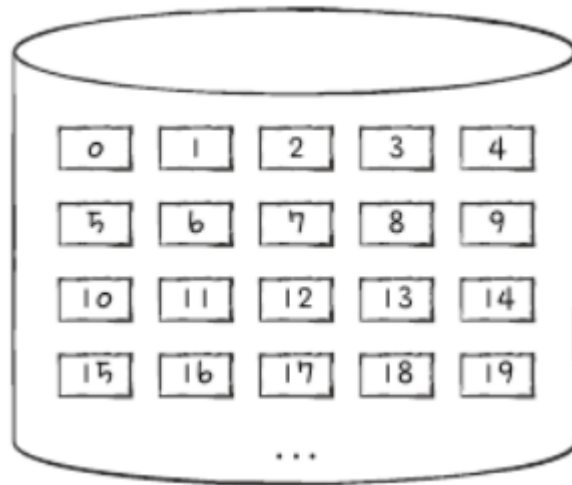
포맷 옵션(O)
☒ 빠른 포맷(Q)

시작(S) 닫기(C)

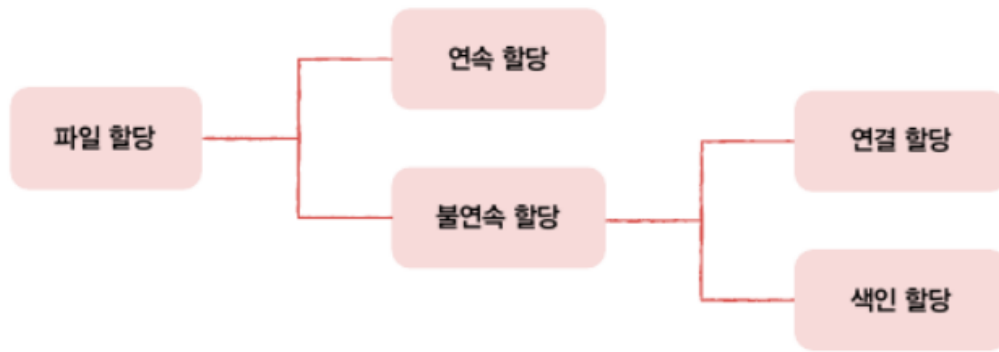
- 포매팅할 때 파일 시스템이 결정
- 파일 시스템에는 여러 종류가 있고, 파티션마다 다른 파일 시스템 설정 가능
- **파티셔닝**과 **포매팅**은 동시에 진행되는 경우가 많음

◎ 파일 할당 방법

- 포매팅까지 끝냈으면, 여기에 사용할 파일을 저장해보자
- 운영체제는 파일과 디렉터리를 **블록** 단위로 읽고 씀
- 하나의 파일이 보조기억장치에 저장될 때에는 하나 이상의 블록에 걸쳐 저장
- 하드 디스크의 가장 작은 저장 단위는 섹터

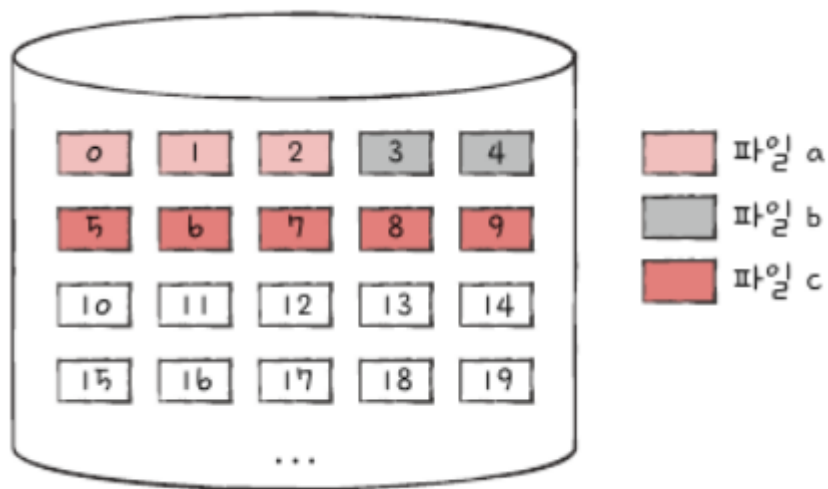


- 것처럼 하드디스크 내 여러 블록이 있다고 가정
 - 블록 안에 적힌 번호는 블록의 위치를 식별하는 주소
 - 이 블록에 사용하는 파일들을 할당
 - 크기가 작은 파일은 적은 수의 블록에 걸쳐 저장
 - 크기가 큰 파일은 여러 블록에 걸쳐 저장
-
- 위와 같은 상황에서 보조기억장치에 할당하는 방법 두 가지
 - **연속 할당**과 **불연속 할당**
 - 불연속 할당에는 연결할당과 색인할당



◎ 연속 할당

- 연속 할당은 가장 단순한 방식
- 보조기억장치 내 연속적인 블록에 파일을 할당하는 방식
- 블록 세 개, 두 개, 다섯 개 차지하는 정도 크기를 가진 파일 a, b, c

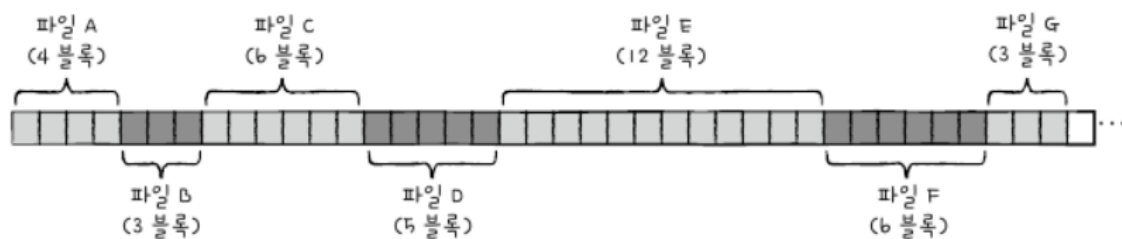


- 연속으로 할당된 파일에 접근하기 위해서는 파일의 첫번째 블록 주소와 블록 단위의 길이만 알면 됨
- 파일 시스템에서는 디렉터리 엔트리에 파일 이름과 더불어 첫 번째 블록 주소와 블록 단위의 길이를 명시

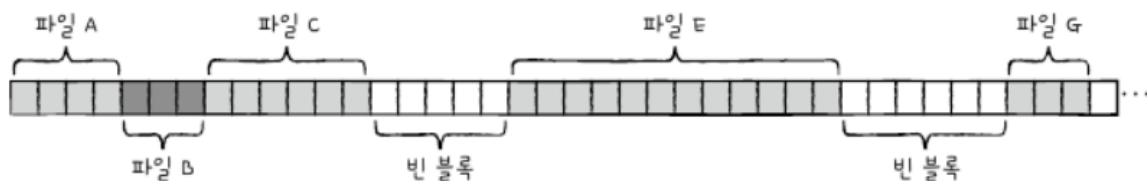
디렉터리

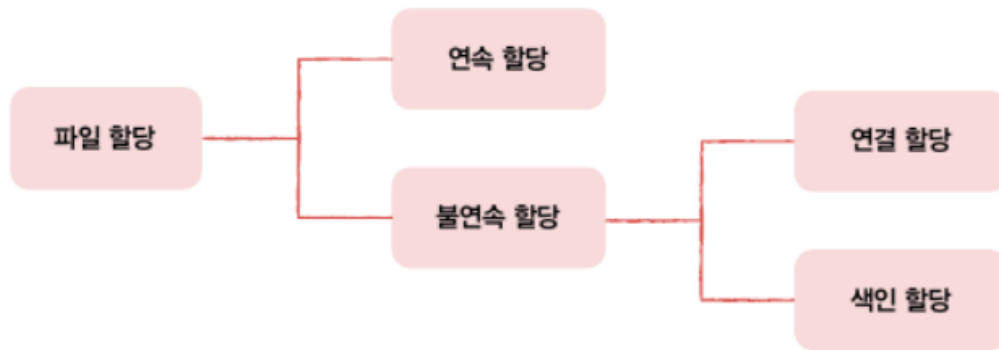
파일 이름	첫 번째 블록 주소	길이
a	2	3
b	5	2
c	10	5

- 연속 할당 방식은 구현이 단순하다는 장점
- 하지만 **외부 단편화**를 야기한다는 치명적인 문제점이 있음



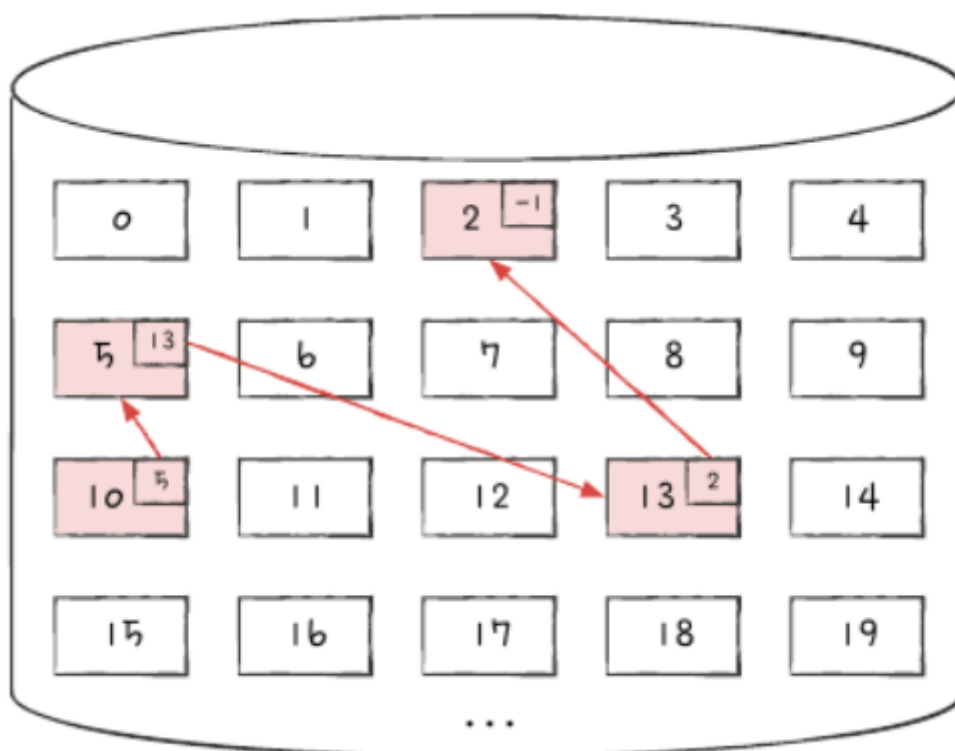
- 파일 D와 F가 삭제되면 할당 가능 블록은 총 11개가 남지만, 불행히도 크기가 블록 7개 이상을 사용하는 파일은 할당할 수 없음





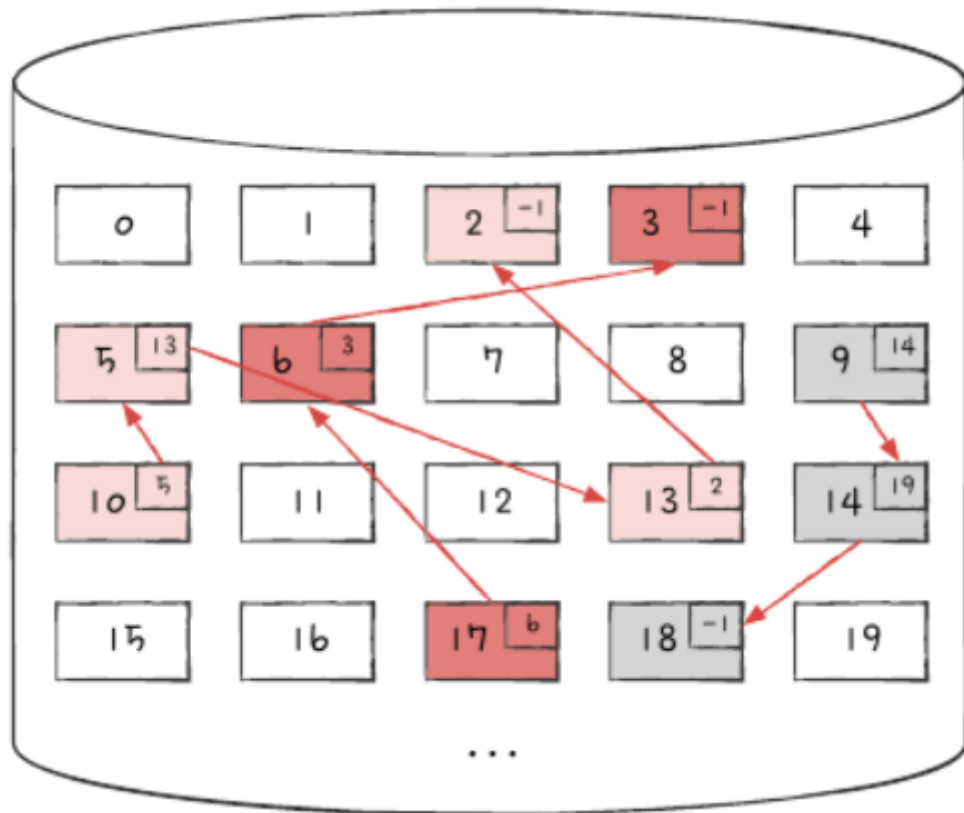
◎ 연결 할당

- 연속 할당의 문제를 해결할 수 있는 방식
- 각 블록의 일부에 다음 블록의 주소를 저장하여 각 블록이 다음 블록을 가리키는 형태로 할당하는 방식
- 파일을 이루는 데이터를 연결 리스트로 관리
- 연결 할당은 불연속 할당의 일종이기에 파일이 여러 블록에 흩어져 저장되어도 무방



- 10번 블록은 5번 블록을, 5번 블록은 13번 블록을 ...

- 파일 시스템에서는 디렉터리 엔트리에 연속 할당과 마찬가지로 파일 이름과 함께 첫 번째 블록 주소와 블록 단위의 길이를 명시



디렉터리

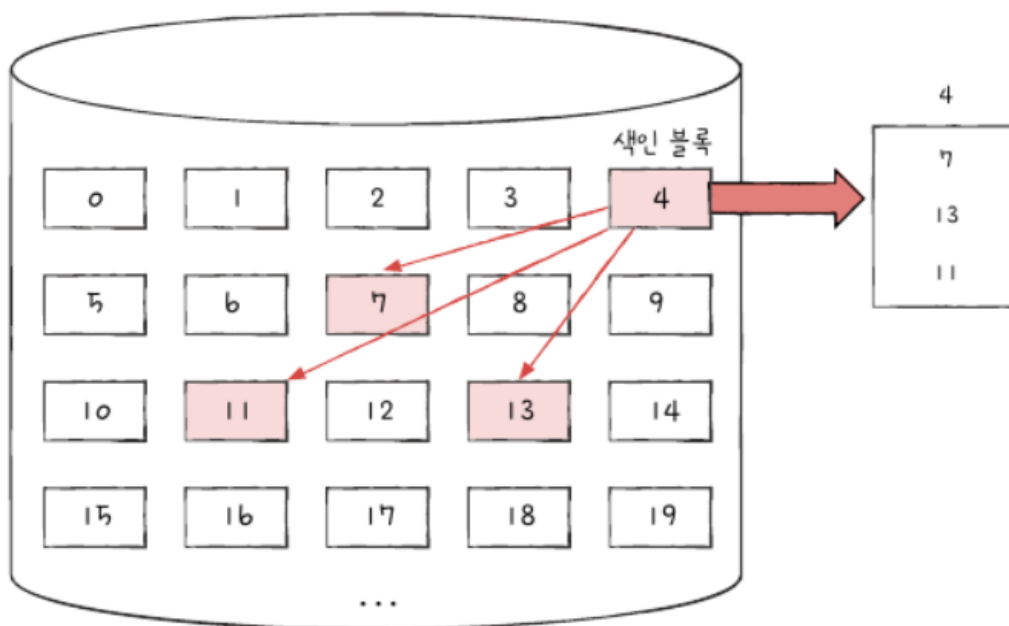
파일 이름	첫 번째 블록 주소	길이
a	10	4
b	9	3
c	17	3

- 디렉터리 엔트리에 첫 번째 블록 주소와 마지막 블록 주소 기록 가능
- **연결 할당**은 외부 단편화 문제를 해결하지만 이 또한 **단점**이 있음

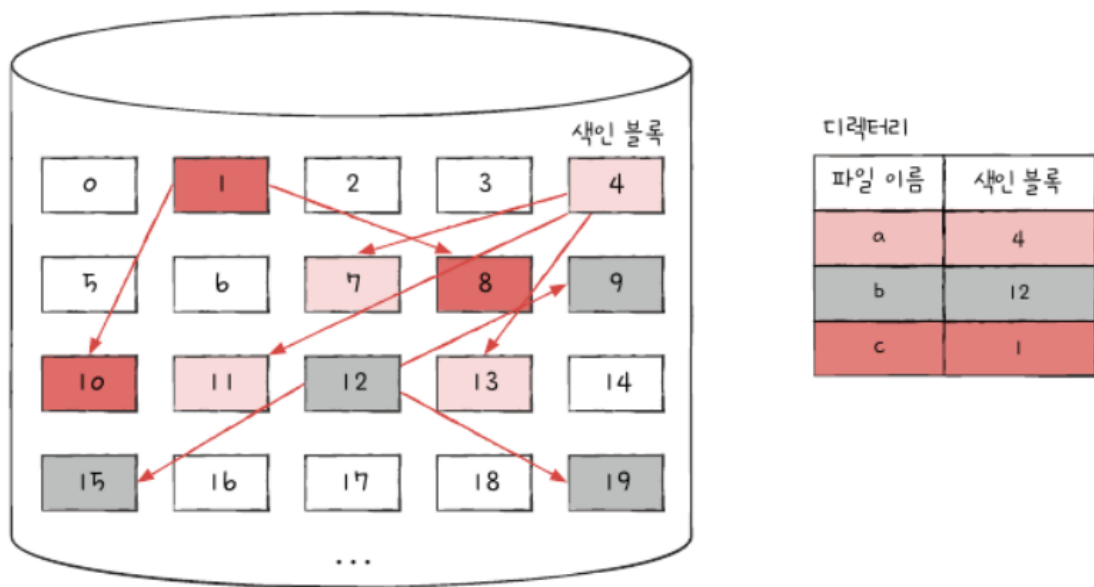
- 첫째, 반드시 첫 번째 블록부터 하나씩 차례대로 읽어야 한다.
- 파일의 중간 부분부터 접근하고 싶어도 반드시 파일의 첫번째 블록부터 접근하여서 하나씩 차례대로 읽어야 한다. 파일 내 임의의 위치에 접근하는 속도, 즉 **임의 접근 속도**가 매우 느림.
- i번째 블록에 접근하기 위해 반드시 첫 번째 블록부터 i번째 블록까지 일일이 순서대로 접근해야 함 → **비효율적**
- 둘째, 하드웨어 고장이나 오류 발생 시 해당 블록 이후 블록은 접근할 수 없다.
- 하나의 블록 안에 파일 데이터와 다음 블록 주소가 모두 포함 → 하나라도 문제 발생 시 그 이후의 블록에 접근할 수 없다.
- 연결 할당을 변형한 대표적인 파일 시스템 → FAT 파일 시스템

◎ 색인 할당

- 연결 할당은 블록 일부에 다음 블록 주소를 표현하는 방식
- 반면 색인 할당은 파일의 모든 블록 주소를 색인 블록이라는 하나의 블록에 모아 관리하는 방식



- 파일 A의 색인 블록은 4번 블록
- 파일 A의 데이터는 7, 13, 11번에 저장되어 있다고 가정
- 색인 할당은 연결 할당과 달리 파일 내 임의의 위치에 접근하기 쉬움
- 파일의 i번째 데이터 블록에 접근하고 싶다면, 색인 블록의 i번째 항목이 가리키는 블록에 접근하면 되기 때문
- 색인 할당을 사용하는 파일 시스템에서는 디렉터리 엔트리에 파일 이름과 더불어 **색인 블록 주소** 명시



- 이러한 색인 할당을 기반으로 만든 파일 시스템 → 유닉스 파일 시스템

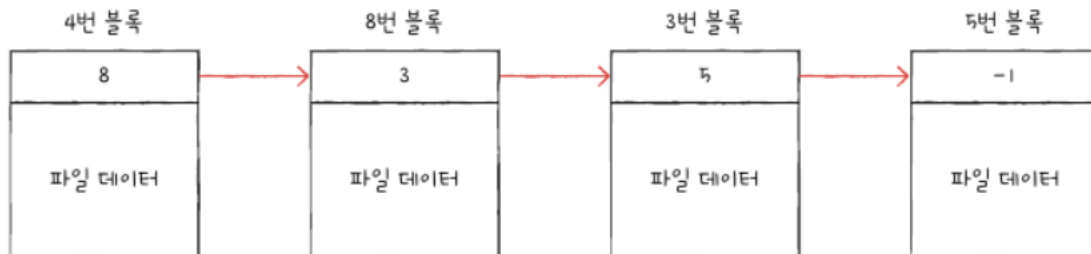
◎ 파일 시스템 살펴보기

- FAT 파일 시스템 - USB 메모리, SD 카드 등의 저용량 저장 장치에서 사용
- 유닉스 파일 시스템 - 유닉스 계열 운영체제에서 사용

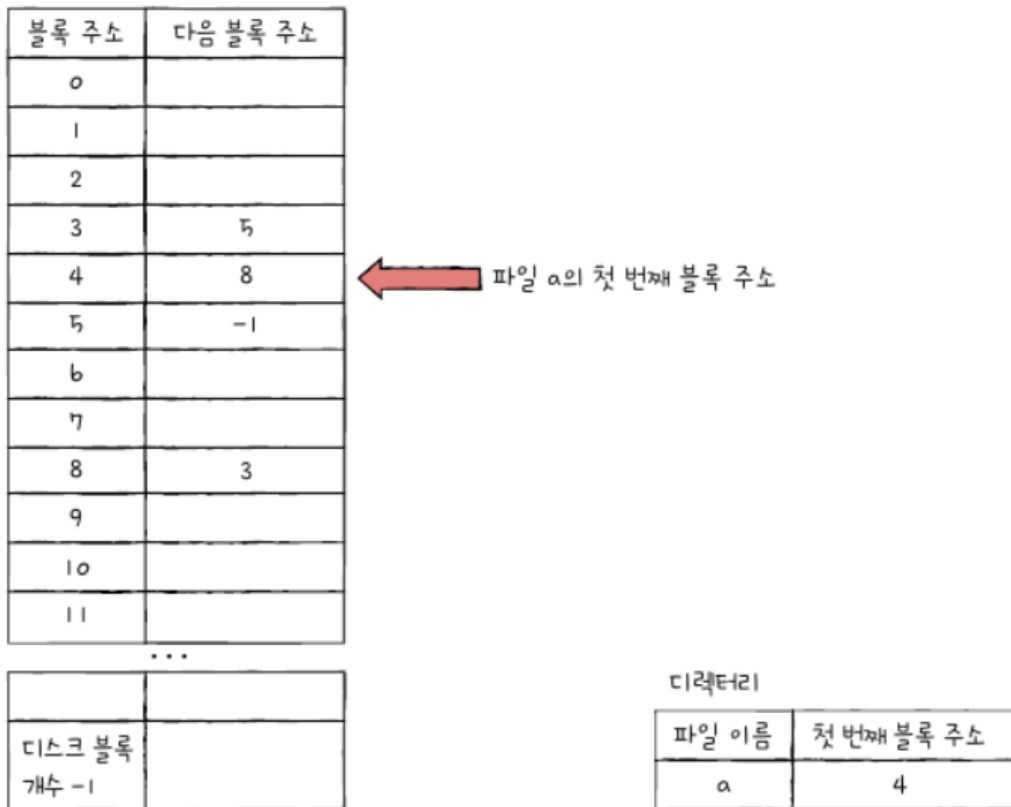
◎ FAT 파일 시스템

- 연결 할당의 단점을 보완한 파일 시스템이 FAT 파일 시스템

- 단점의 근본적 원인 → 블록 안에 다음 블록의 주소를 저장하였기 때문
- 연결 할당을 단순화한 그림



- 아까처럼 연결 할당의 경우 그 다음 블록 주소를 저장하기 때문에 임의 접근 성능이 좋지 않고, 하나의 블록에서 문제 발생 시 접근 불가능
- 하지만 각 블록에 포함된 다음 블록의 주소들을 모아 테이블 형태로 관리하면 앞서 언급한 단점들 상당 부분 해소 → **파일 할당 테이블 (FAT : File Allocation Table)**
- 첫번째 블록만 알면 파일의 데이터가 담긴 모든 블록에 접근 가능 → 디렉터리 엔트리에 는 파일 이름과 더불어 파일의 첫번째 블록 주소가 명시

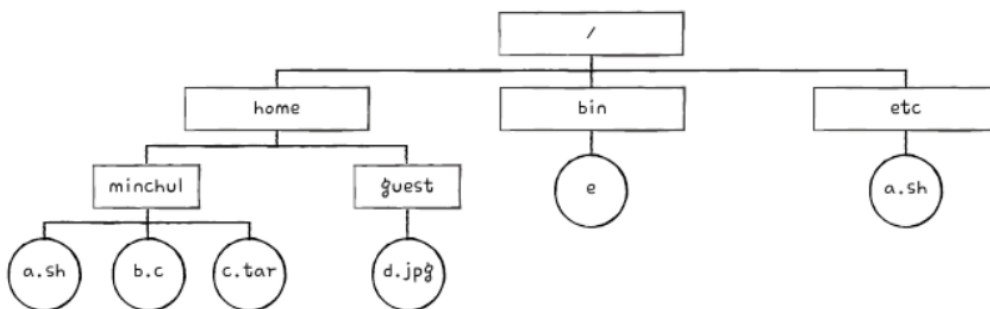


- 더 이상 다음 블록이 없으면 특별한 표시자 표기 (위에서는 -1)
- 이러한 FAT을 이용하는 파일 시스템 → **FAT 파일 시스템**
- 옛날 마이크로소프트 운영체제 MS-DOS에서 사용, 최근까지 저용량 저장 장치(USB 메모리, SD 카드)용 파일 시스템으로 많이 이용
- FAT 파일 시스템은 버전에 따라 → **FAT12, FAT16, FAT32**가 있음 (숫자는 블록을 표현하는 비트 수)
- 윈도우에서는 '블록'이라는 용어 대신 클러스터라는 용어 사용 (FAT 뒤에 오는 숫자는 클러스터를 표현하기 위한 비트)
- FAT 파일 시스템에서 FAT은 파티션의 앞부분에 만들어짐
- 하드디스크의 한 파티션을 FAT 파일 시스템으로 포맷 시 해당 파티션이 다음과 같이 구성됨



- FAT 영역에 FAT가 저장되고, 뒤이어 루트 디렉터리가 저장되는 영역
- 그 뒤에 서브 디렉터리와 파일들을 위한 영역
- FAT는 하드 디스크 파티션의 시작 부분에 있지만, 실행 도중 FAT가 메모리에 캐시될 수 있다. FAT가 메모리에 적재된 채 실행되면 기존 연결 할당보다 다음 블록을 찾는 속도가 매우 빨라짐
- 결과적으로 앞서 설명한 연결 할당 방식보다 임의 접근에도 유리해짐
- FAT가 메모리에 적재된 채 실행되면 임의 접근의 성능이 개선됨
- FAT 파일 시스템의 디렉터리 엔트리
- 파일 이름과 더불어 파일의 첫번째 블록주소가 명시
- FAT 파일 시스템에서의 디렉터리 엔트리에는 파일 속성과 관련한 다양한 정보들이 다음과 같은 형식으로 블록에 저장

파일 이름	확장자	속성	예약 영역	생성 시간	마지막 접근 시간	마지막 수정 시간	시작 블록	파일 크기
-------	-----	----	-------	-------	-----------	-----------	-------	-------

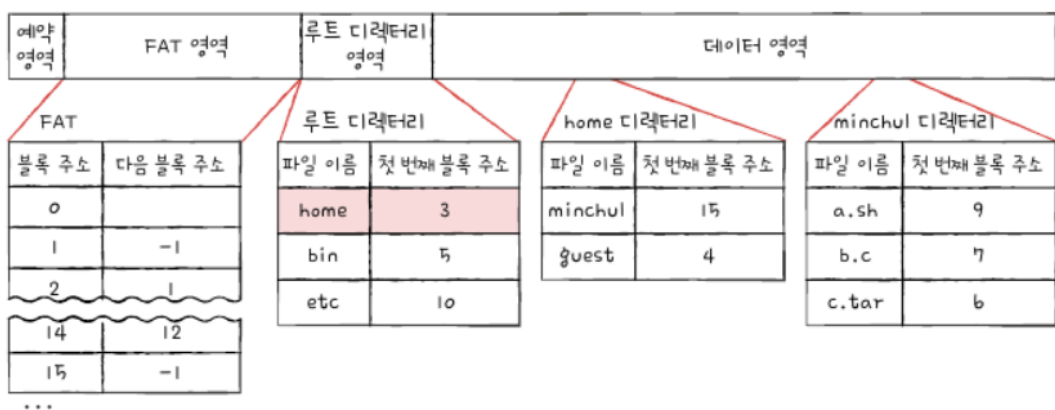


- /home/minchul/a.sh 파일을 읽는 과정

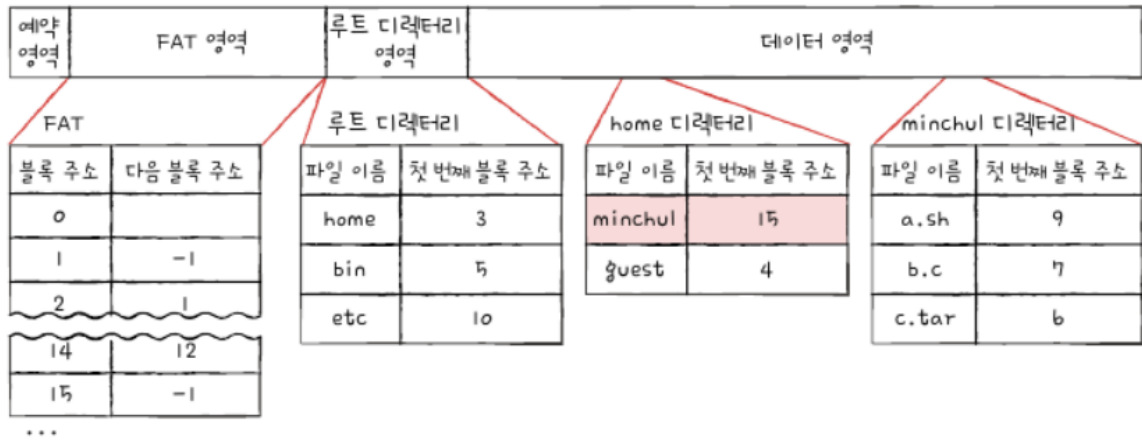
- 이 디렉터리 구조는 파티션 내에 다음 그림과 같이 저장됨
- -1은 FAT 상에서 더 이상의 블록이 없음을 표시하기 위한 표시자



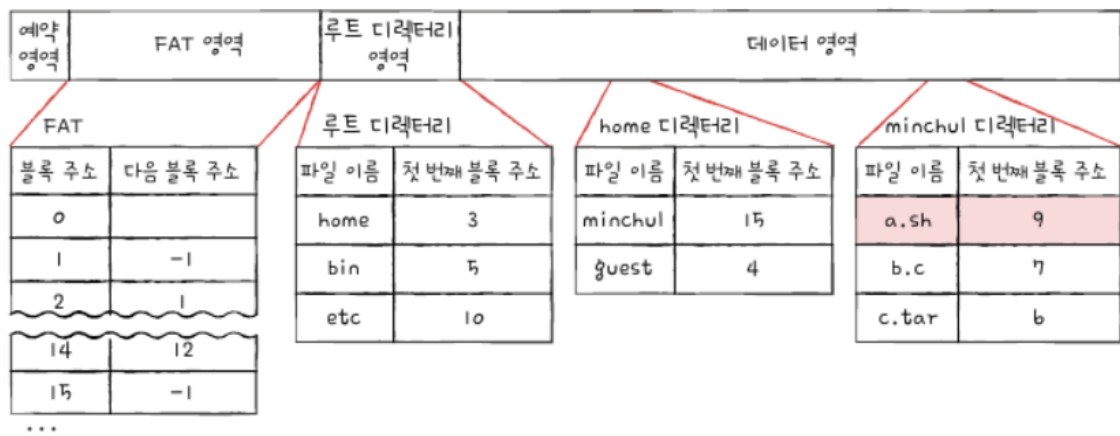
- 루트 디렉터를 보자. home 디렉터리는 3번 블록



- 3번 블록을 읽어 home 디렉터리 내용을 보자. minchul 디렉터리는 15번 블록



- 15번 블록을 읽어 minchul 디렉터리 내용을 보자. a.sh 파일의 첫 번째 블록 주소가 9번 블록



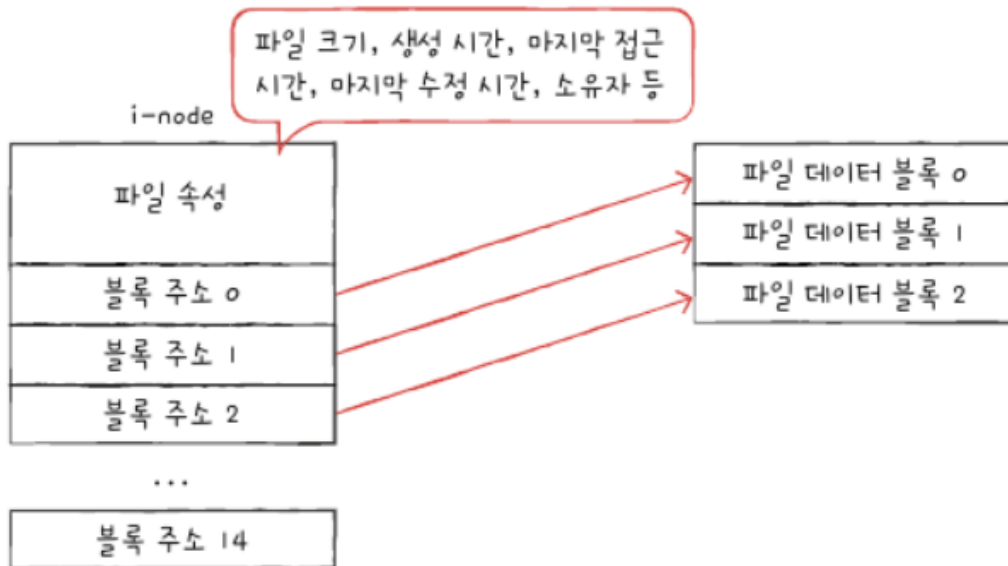
- FAT을 보면 a.sh 파일은 9, 8, 11, 13번 블록 순서로 저장되어 있다는 것을 알 수 있음



- 파일 시스템은 /home/minchul/a.sh를 읽기 위해 9번, 8번, 11번, 13번 블록에 접근

◎ 유닉스 파일 시스템

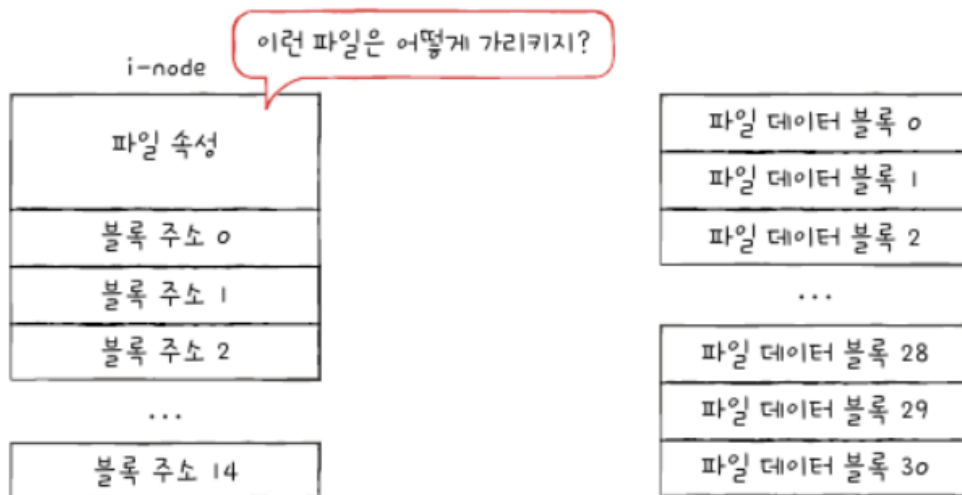
- 색인 할당 기반의 유닉스 파일 시스템
- 색인 할당 : 색인 블록으로 파일 데이터 블록들을 찾는 방식
- 유닉스 파일 시스템의 색인 블록 → **i-node (index-node)**



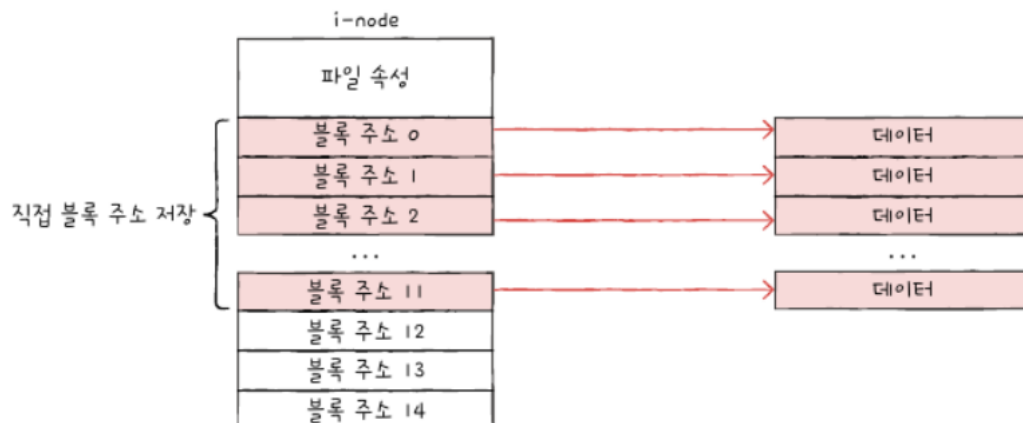
- FAT 파일 시스템에는 파일 속성 정보가 디렉터리 엔트리에 표현
- 유닉스 파일 시스템에서 파일 속성 정보가 i-node에 표현
- 유닉스 파일 시스템에는 파일마다 이러한 i-node가 있고 i-node마다 번호가 부여
- i-node들은 다음과 같이 파티션 내 특정 영역에 모여 있음



- i-node 영역에 i-node들이 있고, 데이터 영역에 디렉터리와 파일들이 있음
- 여기서 문제가 있다.
- i-node의 크기는 유한함
- i-node 하나는 기본적으로 열다섯 개의 블록 주소를 저장할 수 있음

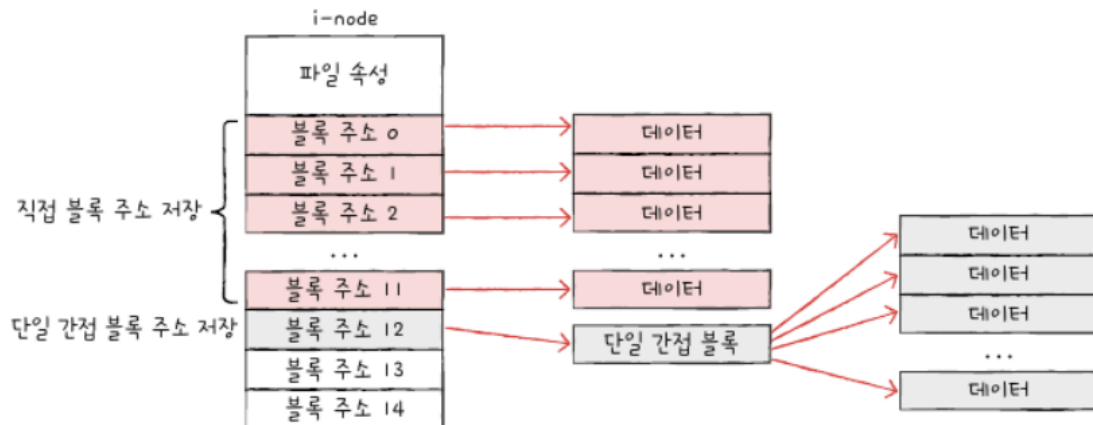


- 하지만 블록을 20, 30개 차지하는 큰 파일 → i-node 하나만으로 파일의 데이터 블록을 모두 가리킬 수 없음
- 유닉스 파일 시스템은 이러한 문제를 다음과 같이 해결
- 첫째, 블록 주소 중 열두 개에는 직접 블록 주소를 저장



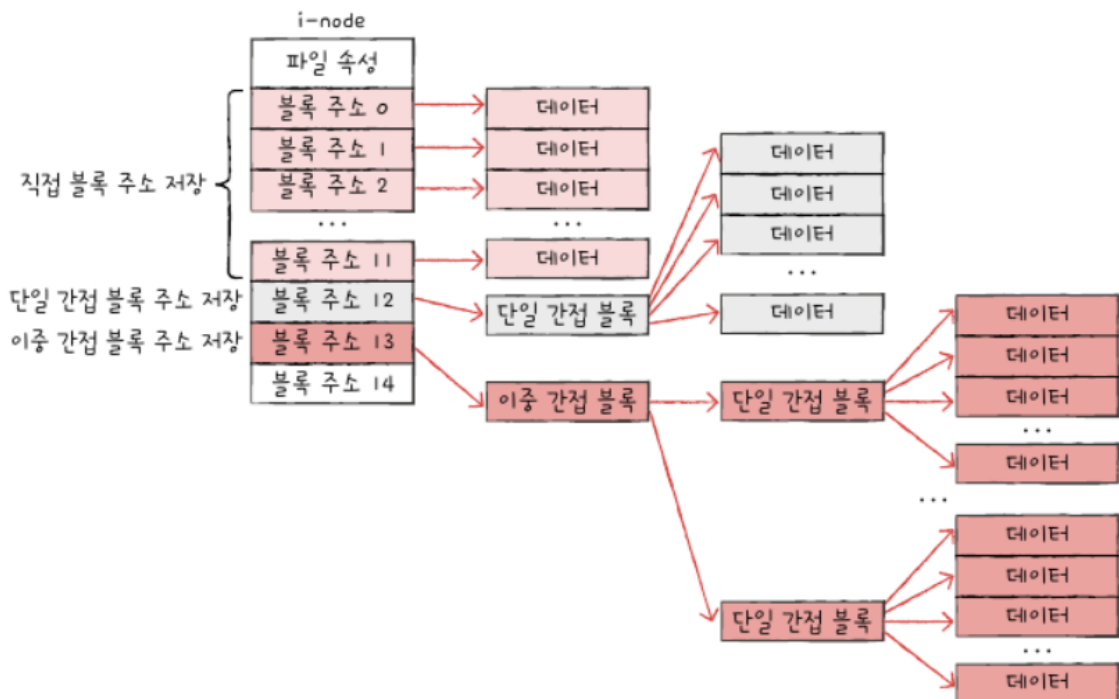
- i-node가 가리킬 수 있는 열 다섯개의 블록 주소 중 처음 열 두개에는 파일 데이터가 저장된 블록 주소가 직접적으로 명시 → **직접 블록**
- i-node의 열 두개 주소는 직접 블록 주소 저장
- 이것만으로 파일 데이터 블록 모두 가리킬 수 있으면 추가 작업 X

- 둘째, ‘첫째’ 내용으로 충분하지 않다면 열 세번째 주소에 단일 간접 블록 주소를 저장



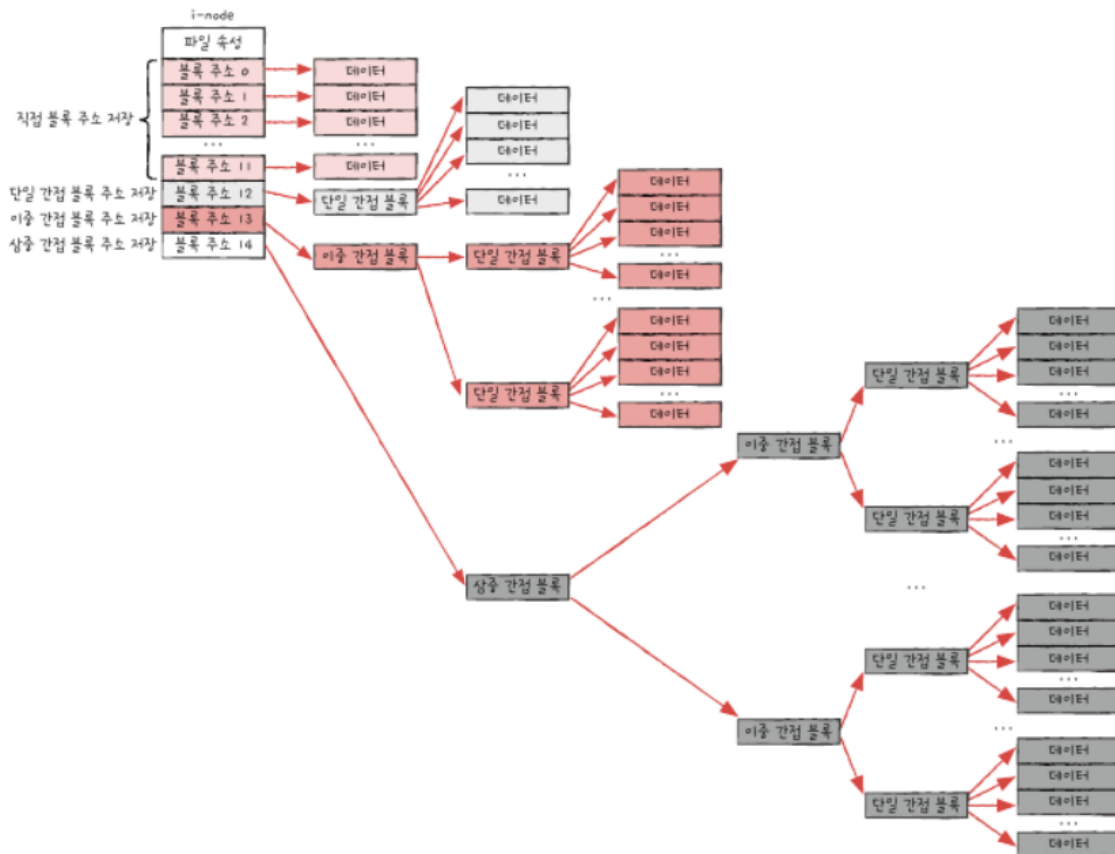
- 열 세번째 블록 주소는 단일 간접 블록의 주소를 저장.
- **단일 간접 블록** : 파일 데이터가 저장된 블록이 아닌 파일 데이터를 저장한 블록 주소가 저장된 블록

- 셋째, ‘둘째’ 내용으로 충분하지 않다면 열 네번째 주소에 이중 간접 블록을 저장

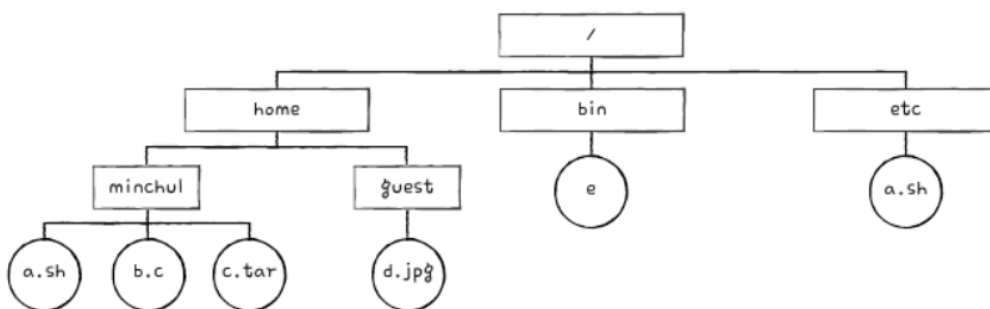


- 이중 간접 블록 : 데이터 블록 주소를 저장하는 블록 주소가 저장된 블록 → 단일 간접 블록들의 주소를 저장하는 블록

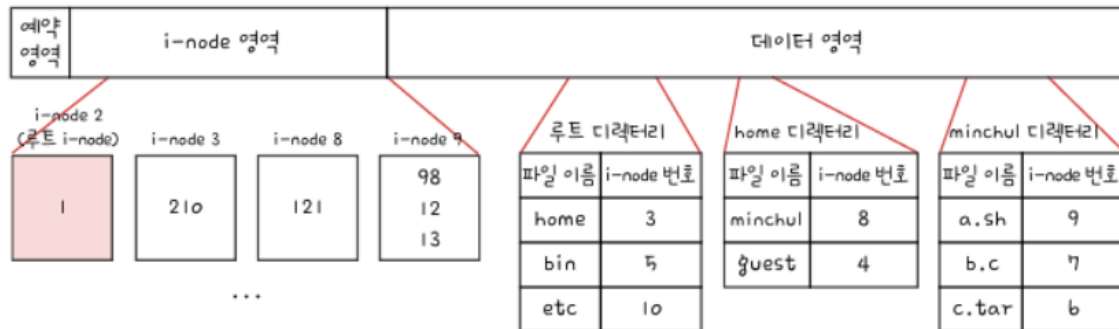
- 넷째, ‘셋째’ 내용으로 충분하지 않다면 열다섯번째 주소에 삼중 간접 블록 주소를 저장



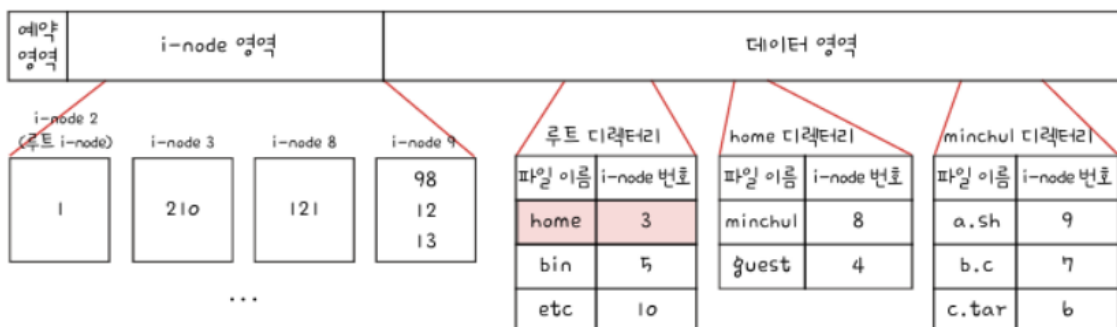
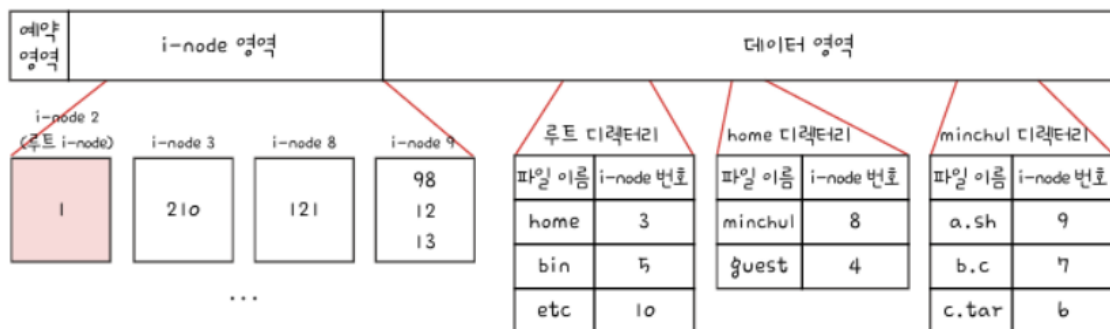
- i-node만 알면 파일 속성 뿐만 아니라 파일 크기가 크더라도 파일 데이터를 모두 가리킬 수 있음

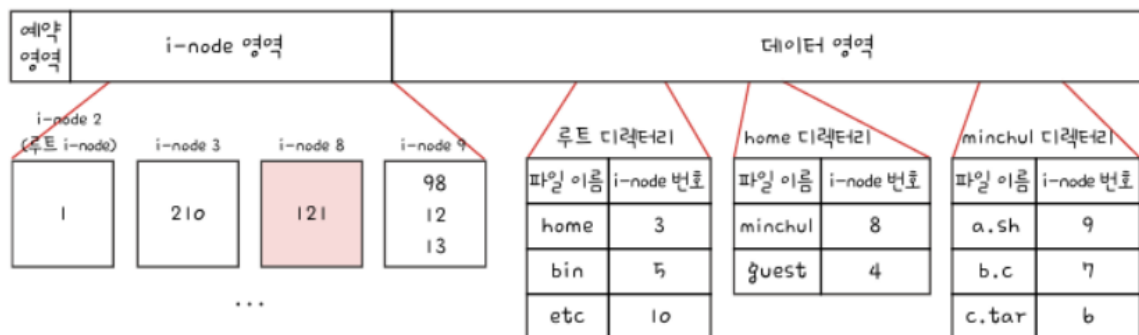
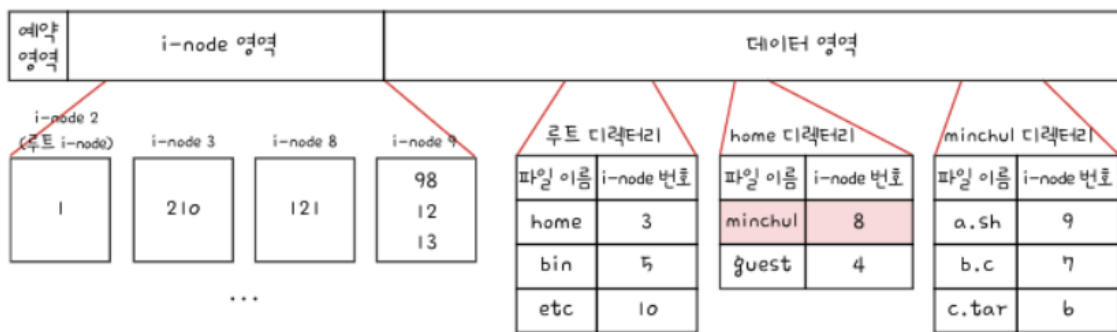
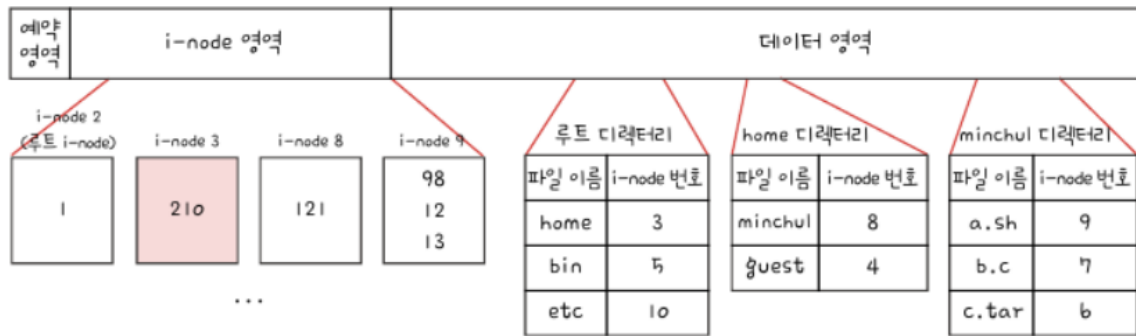


- 유닉스 파일 시스템에서 /home/minchul/a.sh 파일을 읽는 과정

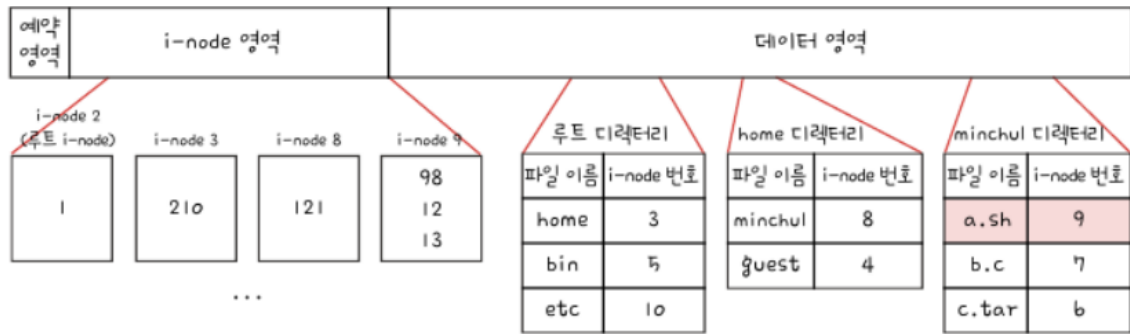


- 루트 디렉터리 위치부터 찾을
- 루트 디렉터리의 i-node (2번 i-node)

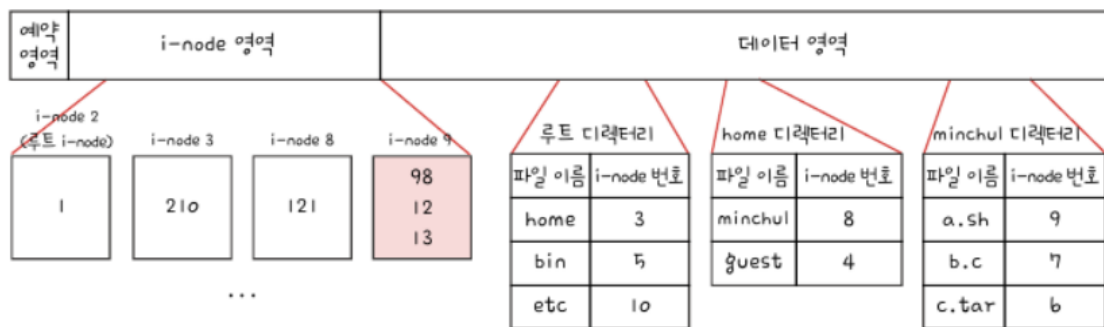




- 121번 블록 이동



- 9번 i-node에 접근해 파일 a.sh 위치 파악 → 98번, 12번, 13번 블록에 있음



◎ 그 외 다른 파일 시스템

- NT 파일 시스템 (NTFS) → 윈도우 운영체제
- ext 파일 시스템 → 리눅스 운영체제

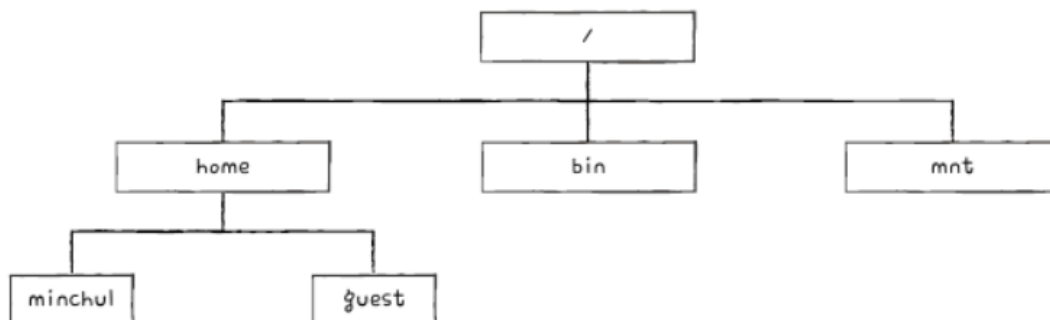
◎ 저널링 파일 시스템

- 컴퓨터 작업 도중 전원 나가거나 치명적 오류로 컴퓨터가 강제 종료되어 버린 상황
- 파일 시스템 변경 도중 이러한 상황(시스템 크래시)이 발생하면 파일 시스템이 훼손될 수 있음
- 저널링 파일 시스템 있기 전 이런 상황 발생 시 부팅 직후 파일 시스템 검사하고 복구 프로그램 실행 → 파일 시스템 내의 모든 블록에 대해 파일 시스템 검사하기 때문에 매우 오래 걸린다
- 저널링 기법을 이용하는 **저널링 파일 시스템**

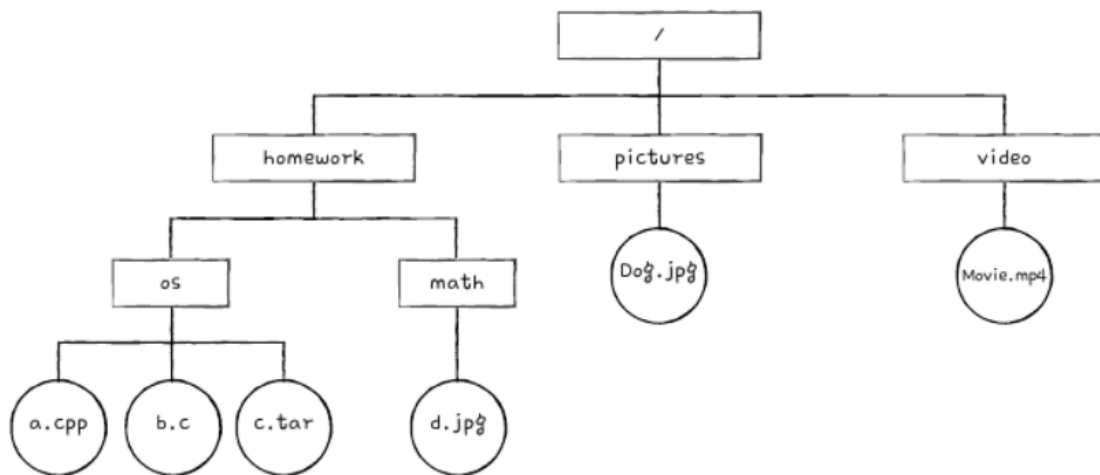
- **저널링 기법** : 작업 로그를 통해 시스템 크래시 발생 시 빠르게 복구하기 위한 방법
 1. 작업 직전 파티션의 로그 영역에 수행하는 작업(변경 사항)에 대한 로그를 남긴다.
 2. 로그를 남긴 후 작업을 수행한다
 3. 작업이 끝났다면 로그를 삭제
- 현대 대부분의 파일 시스템은 이러한 저널링 기능을 지원

◎ 마운트

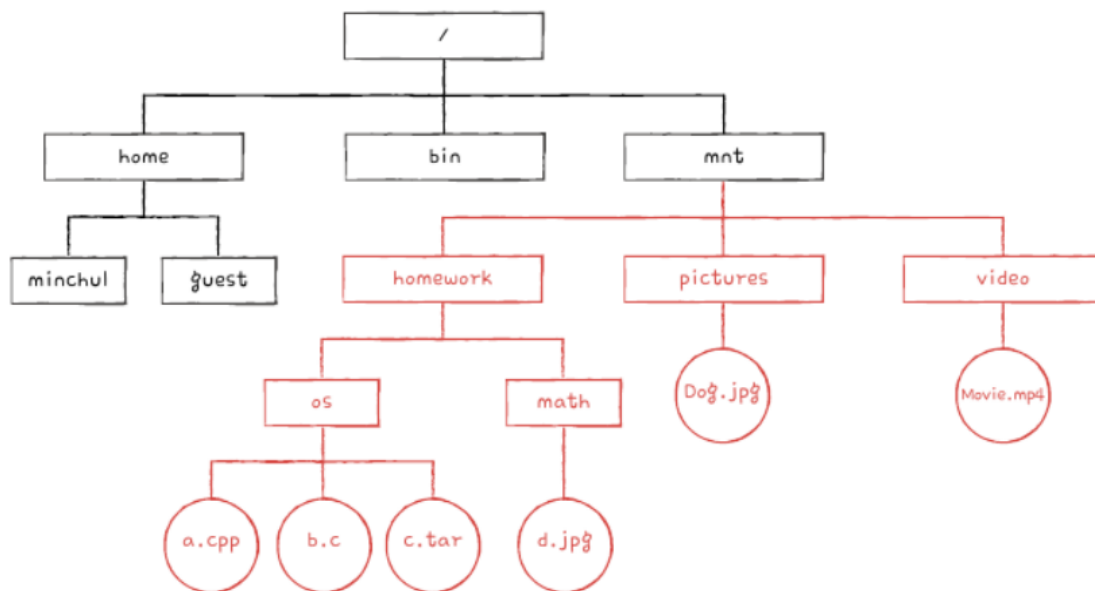
- 한 저장장치의 파일 시스템에서 다른 저장 장치의 파일 시스템에 접근할 수 있도록 파일 시스템을 편입 시키는 작업을 의미
- 컴퓨터 디렉터리 구조 예시



- USB 메모리 디렉터리 구조 예시



- 마운트



- 유닉스, 리눅스와 같은 운영체제에서 다양한 저장 장치를 컴퓨터에 연결할 때 mount 명령어로 빈번하게 마운트