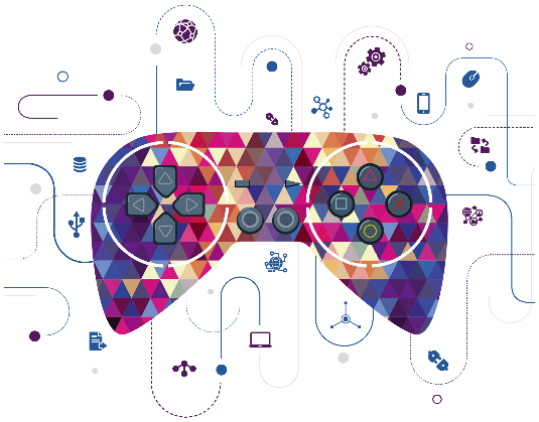


그림으로 배우는 구조와 원리

운영체제 개정 3판

Chatpter 10
파일 관리

- 파일 시스템의 기능과 구조, 파일
- 파일을 관리하는 디렉터리 시스템
- 파일의 디스크 할당 방법
- 파일의 보호

Section 01 파일 시스템과 파일(1. 파일 시스템의 개념)

■ 파일 시스템의 개념

- 논리적 저장 단위인 파일을 정의하고 메모리에 매핑시키는 기능을 제공
- 파일을 구성하고 데이터 액세스를 관리하므로, 파일 시스템은 데이터를 실제로 저장하는 파일과 이를 계층적으로 연결하는 디렉터리로 구성
- 디스크 저장소를 관리(저장 공간 할당)하는 것과 관련이 많고, 메인 메모리와 다른 매체에 저장된 파일 데이터 액세스도 포함
- 사용자가 파일을 생성·수정·삭제할 수 있도록 지원하며, 파일을 각 응용 프로그램에 가장 적합한 구조로 구성할 수 있도록 지원
- 파일 읽기, 쓰기, 실행 등을 다양하게 액세스하도록 제어된 방법도 제공
- 파일 시스템 설계
  - 사용자 수, 사용자당 평균 파일 수와 크기 등 사용자 정보가 필요하고, 해당 시스템에서 실행할 응용 프로그램의 특성을 이해하여 적합한 파일 구성과 디렉터리 구조 결정해야 함

2. 파일 시스템의 기능

파일 시스템의 기능

- 파일 구성 : 사용자가 각 응용 업무에 적합하게 파일을 구성할 수 있게 함
- 파일 관리 : 파일의 생성, 수정, 저장, 참조, 제거, 보호 기능을 제공하며, 파일을 공유하는 다양한 액세스 제어 방법을 제공
- 보조 메모리 관리 : 다양한 형태의 저장장치에 입출력 지원, 2차 저장장치에 파일 저장 할 공간 할당
- 파일 무결성 보장 : 파일에 저장된 정보를 손상하지 않도록 보장
- 파일 액세스 방법 제공 : 저장된 데이터(파일) 읽기, 쓰기, 실행, 이들을 여러 형태로 조합한 다양한 액세스 제어 방법을 제공
- 장치 독립성 유지 : 물리적 장치 이름을 사용하는 대신 기호화된 이름(예를 들어m, yDirectory:myFile.txt)으로 자신의 파일을 참조하여 장치와 독립성 유지
- 파일 백업과 복구 : 사고로 정보를 손실하거나 고의로 손상하는 일을 방지하려고 데이터 사본(중복)을 생성하는 백업backup과 손상된 데이터를 복구recovery할 수 있는 기능
- 파일 보호 : 정보를 암호화하고 해독할 수 있는 기능을 제공하여 정보의 안전과 비밀 보장
- 정보 전송 : 사용자가 파일 간에 정보 전송 명령을 내릴 수 있게 한다.

2. 파일 시스템의 기능

■ 파일 시스템의 목적

- 다양한 형태의 저장장치에 입출력을 지원
- 데이터의 보호와 처리율 향상 위해 성능 최적화
- 파일 관리 요소

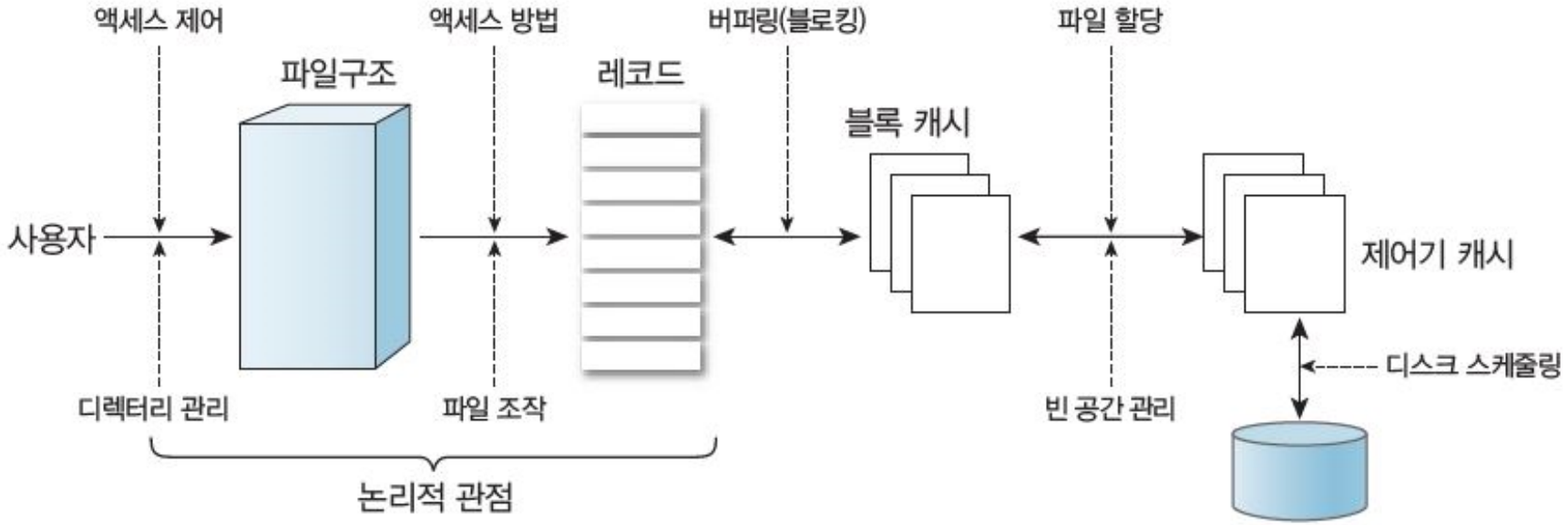


그림 10-1 **파일 관리 요소**

3. 파일 시스템의 구조

■ 파일 시스템의 구분

- 파일의 개념, 속성, 디렉터리 구조, 파일에 허용하는 연산 등을 정의하는 논리적 파일
- 실제 디스크에 이런 논리 파일 시스템을 매핑
- 파일 시스템의 구성

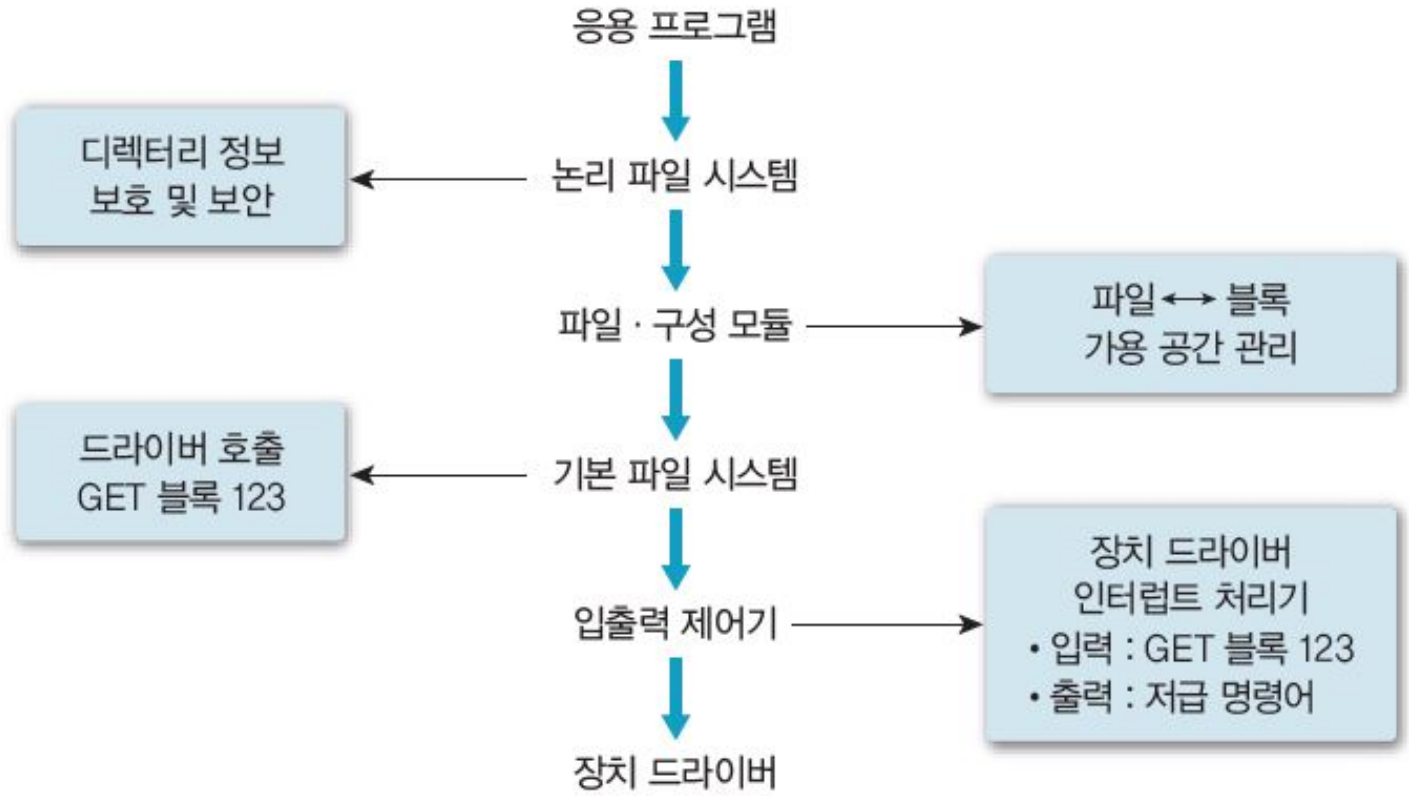


그림 10-2 **계층적 파일 시스템**

4. 파일 시스템의 관리

■ 블록 : 메모리와 디스크 간 전송 단위

- 섹터 하나 이상으로 구성되며, 블록을 할당하는 방법은 사용하는 운영체제에 따라 다름
- 운영체제는 파일에 속하는 정보를 블록에 유지하며, 0에서 시작하는 연속 정수로 일부 최대 수에 달하는 논리 블록 번호 사용
  - 논리 블록 번호는 물리적 디스크 주소(트랙, 실린더, 표면, 섹터)로 변환
  - 변환은 장치 드라이버가 수행하며, 디스크 제어기에 낮은 수준의 하드웨어 명령어로 전달
- 논리적 파일을 물리적 파일로 매핑하는 과정

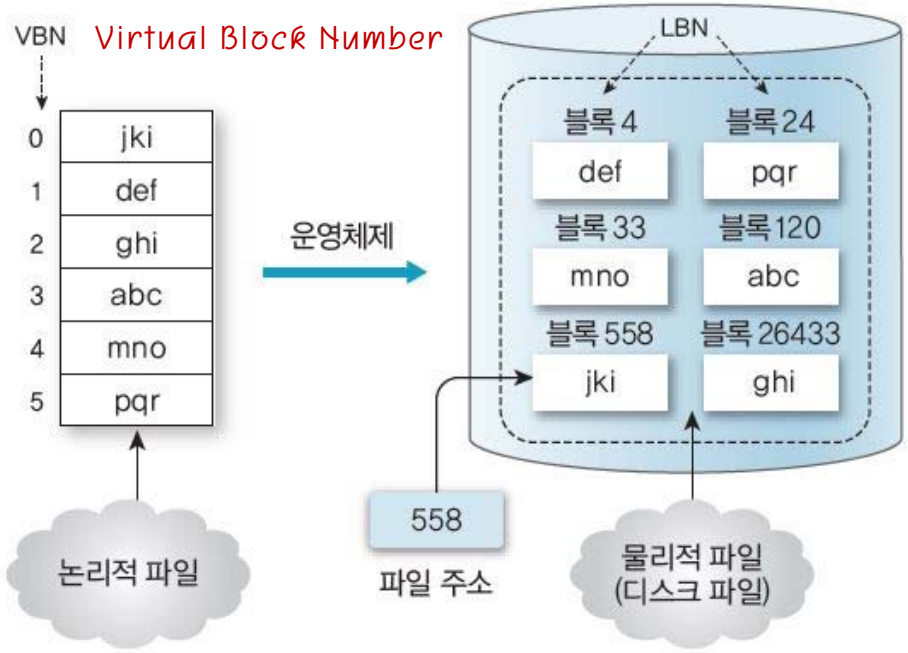


그림 10-3 **논리적 파일을 물리적 파일로 매핑**

4. 파일 시스템의 관리

메타데이터metadata

- 데이터와 디렉터리 외에도 파일 시스템 크기, 새로운 데이터가 이미 사용하는 블록을 덮어쓰지 않도록 저장장치의 가용 블록(공간), 정보(위치), 루트 데이터 위치와 파일 소유자, 크기, 파일을 마지막으로 수정(액세스)한 시간 등 여러 데이터 정보
- 파일 이름은 보통 메타데이터의 일부로 고려하지 않음
- 비어 있는 공간 비트맵, 가용 블록 정보와 불량 섹터 정보 포함
- 데이터와 동일한 방법으로 처리
- 사용자가 직접 수정할 수 없으므로 파일 시스템의 무결성 유지

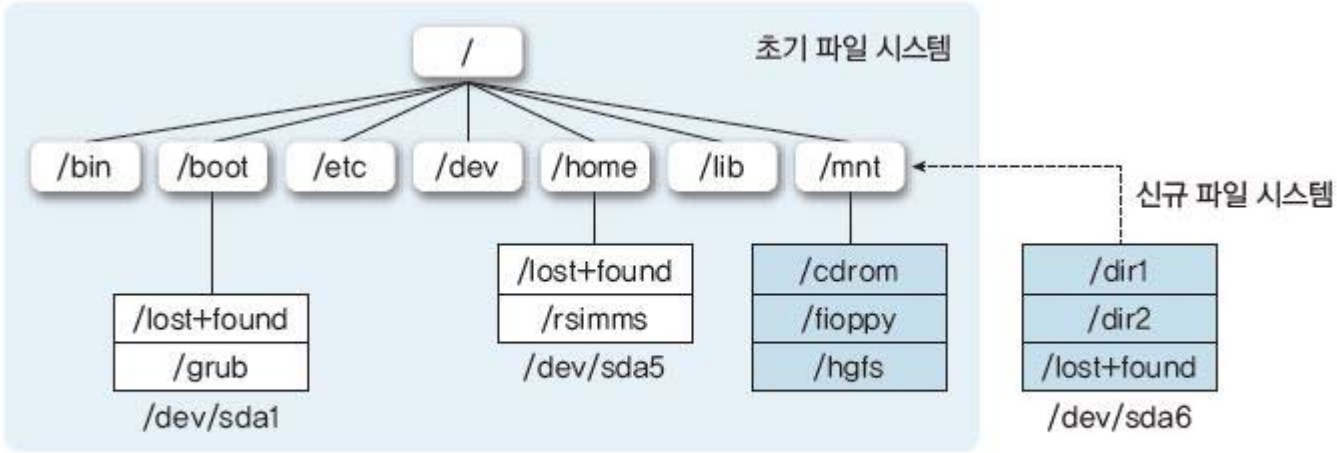
4. 파일 시스템의 관리

마운팅

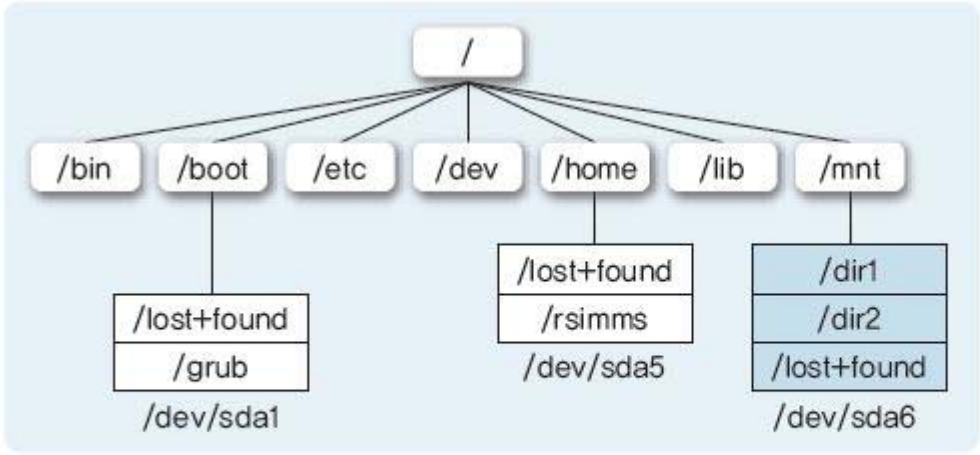
- 시스템에서 초기에 제공하는 파일 시스템, 즉 부팅과 동시에 영구적으로 제공하는 파일 native file의 일부가 아닌 다른 정보에도 액세스해야 할 때 새로운 파일 시스템을 사용하려면 파일 시스템을 미리 제공된 파일 시스템의 디렉터리에 설치하는 절차
- 운영체제는 여러 파일 시스템을 마운트할 수 있는 기능 제공
- 여러 파일 시스템을 단일 파일 시스템에서 식별할 수 있는 파일 집합으로 결합, 운영체제에 마운트하려는 파일 시스템의 저장 위치(장치 이름)와 새로운 파일 시스템의 설치(올려놓을) 지점, 즉 마운트 포인트를 제공하면 됨
- 파일 시스템이 마운트되어 있으면, 사용자는 해당 파일 시스템이 다시 언마운트 될 때까지 마운트 지점에 있는 디렉터리(cdrom 등) 내용을 사용할 수 없음
- 파일 시스템은 마운트 테이블을 사용하여 마운트된 디렉터리들을 관리
- 마운트 테이블은 마운트 지점 경로명, 마운트 된 각 파일 시스템 저장하는 장치 정보 포함

4. 파일 시스템의 관리

- 마운팅 과정



(a) 마운팅 전



(b) 마운팅 후

그림 10-4 **마운팅 전후 파일 시스템**

5. 파일의 개념과 구성

■ 파일의 개념

- 프로그램과 데이터 등 정보의 모음(집합)
- 사용자에게 프로그램, 데이터는 다른 개체이나, 파일 관리 시스템은 동일하게 파일로 처리
- 텍스트처럼 형태가 자유롭거나 엄격하게 제한 가능, 사용 목적에 따라 구조 특별 가능
- 파일 내용은 운영체제가 물리적 장치에 저장.
- 사용자 관점에서 파일은 논리적으로 저장되는 기본 단위로, 프로그램이나 데이터 될 수 있음. 이 논리적 파일을 실제 저장장치에 매핑시키는 작업은 운영체제가 담당
- 파일의 세분화

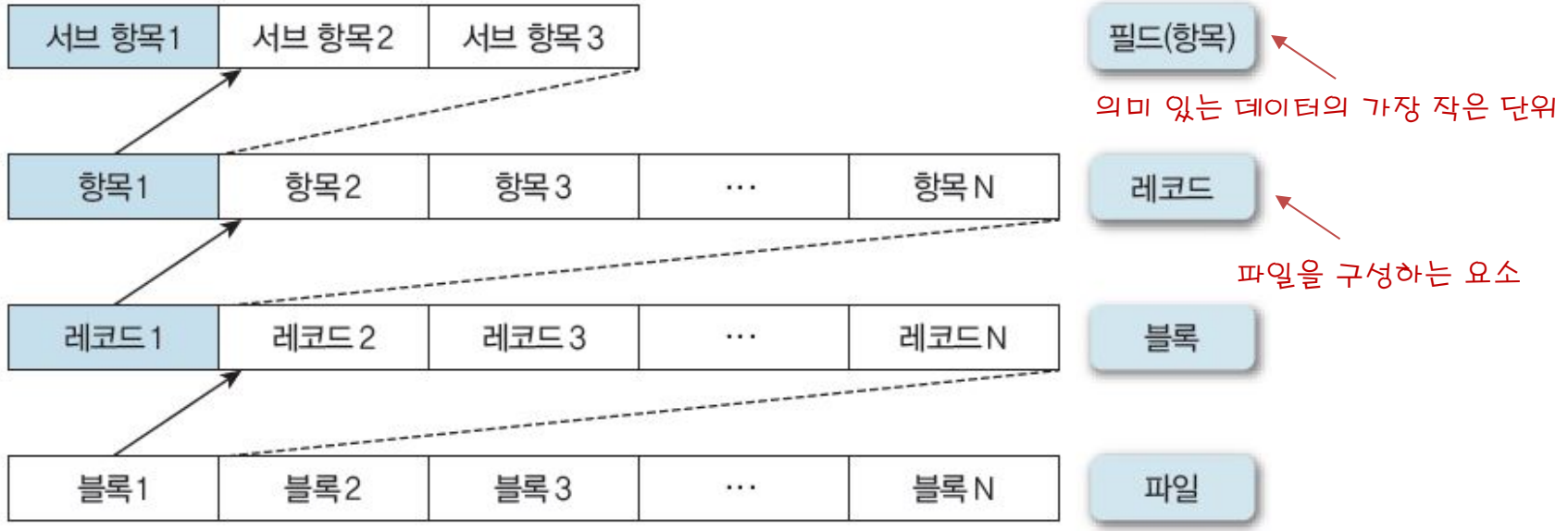


그림 10-5 필드(항목), 레코드, 블록, 파일의 관계

6. 파일의 이름 명명

파일의 이름

- 디렉터리의 루트에서 위치까지 경로 포함
- 특정 파일의 경로명은 디렉터리 이름과 파일 이름으로 구성.
- 모든 경로명을 표시하여 파일을 참조하는 절대 경로가 너무 길어 불편할 때는 작업 디렉터리에서 상대적 위치를 지정하는 상대 경로로 참조 가능

6. 파일의 이름 명명

- 디렉터리 계층 구조와 경로명

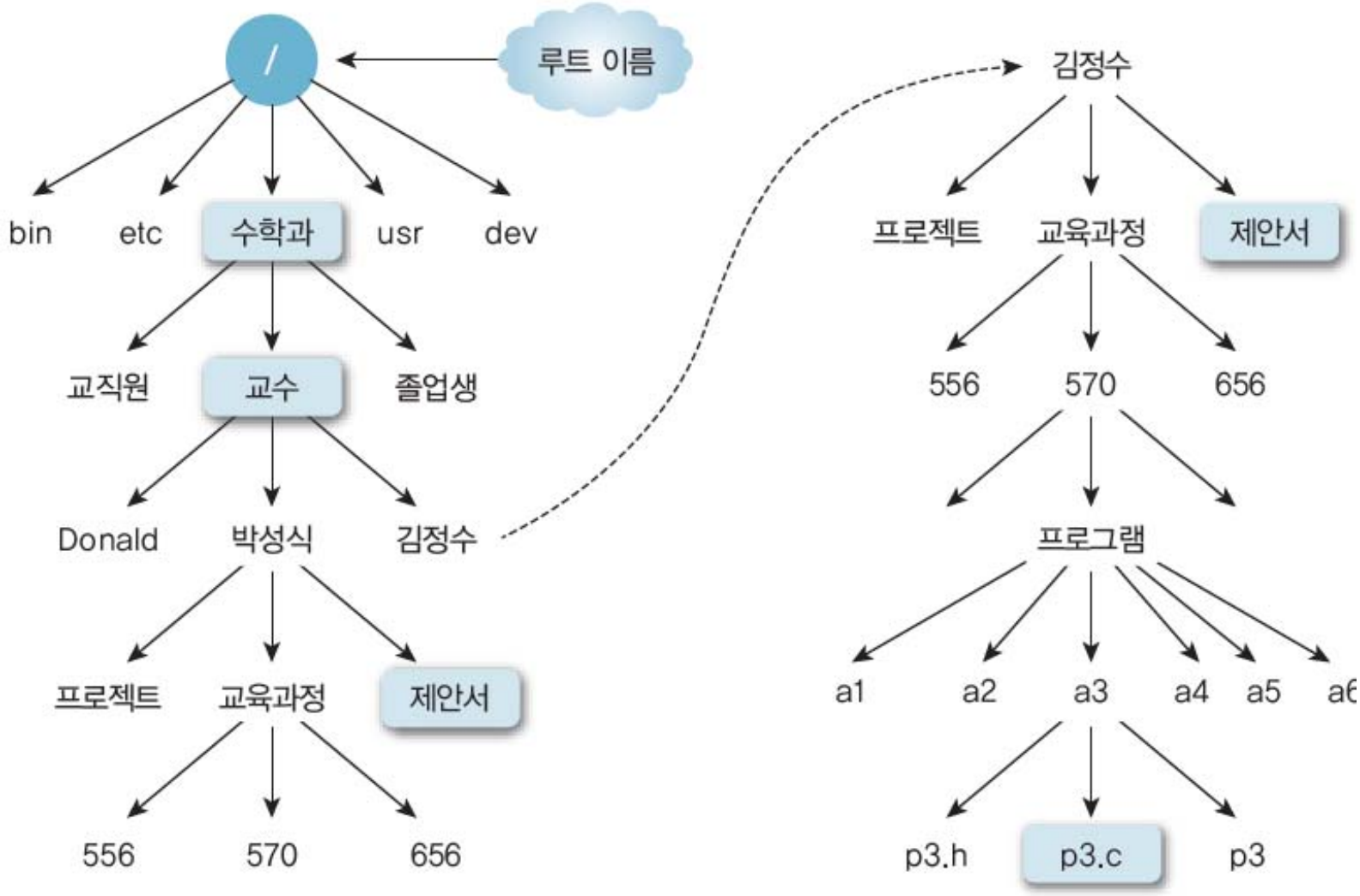


그림 10-6 디렉터리 계층 구조와 경로명

7. 파일의 속성

■ 파일의 속성

- 시스템이 파일을 관리하는 데 필요한 정보
- 대부분 속성을 포함하는 파일 제어 블록은 디스크에 저장, 파일 정보는 메인 메모리에 유지하여 파일을 열 때 탐색 시간 줄임
- 파일 속성은 파일 시스템에 따라 구성 다름
- 파일 속성의 일반적 구성

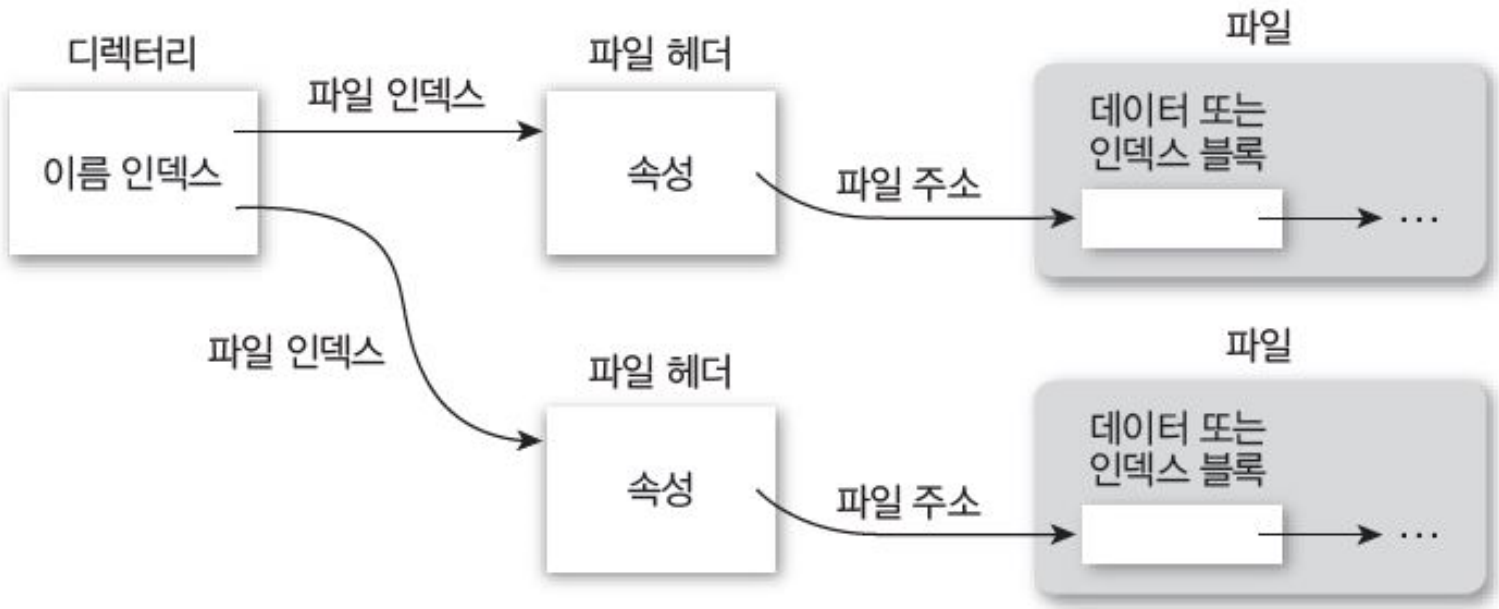


그림 10-7 **파일의 속성**

디렉터리에는 파일 이름이 파일 인덱스한 매핑과 여러 개의 파일이 들어 있음
각 파일에 있는 파일 헤더가 파일 속성과 내용 저장, 유닉스에서는 이 파일 헤더를
i 노드 라고 함

8. 파일의 유형

파일의 유형

- 파일의 유형으로 파일의 내부 구조 형태 짐작 가능
- 운영체제는 파일 시스템이 지원 가능한 파일 구조 정의, 해당 파일을 다룰 수 있는 특별한 연산 기능 제공
- 파일의 세 가지 유형
  - 일반(정규) 파일
    - 가장 일반적인 파일과 데이터를 포함하는 데 사용, 텍스트나 이진 형태
  - 디렉터리 파일
    - 모든 유형의 파일에 액세스할 수 있는 정보 포함하나, 실제 파일 데이터는 포함하지 않음
  - 특수 파일
    - 시스템 장치를 정의하거나 프로세스로 생성한 임시 파일로 파이프라고 하는 FIFO(선입선출), 블록, 문자 이에 해당
- 운영체제가 여러 파일의 구조를 지원하면 크기가 커져 복잡해진다는 단점
  - 너무 적은 파일구조를 지원하면 프로그래밍 어려움
- 파일 이름에 마침표를 넣어서 구분하여 파일 유형 표현
  - 파일 이름은 크게 순수 이름과 확장자로 구성

8. 파일의 유형

- 파일 유형과 확장자 예

표 10-1 **파일 유형과 확장자 예**

파일 유형	확장자	기능
실행 가능	exe, com, bin 등	이진 수행 가능 프로그램
소스 코드	c, p, pas, f77, asm, a	다양한 언어로 된 소스 코드
배치	bat, sh	명령어 해석기에서 명령
문서	txt, doc	텍스트 데이터, 서류
워드 프로세서	wod, hwp, doc, 기타	다양한 워드 프로세서 형식
라이브러리	lib, a, DLL	프로그래머들이 사용하는 라이브러리 루틴
백업, 보관	arc, zip, tar	관련된 파일을 하나로 묶어서 보관하는 것으로 저장용 압축도 가능

9. 파일의 연산

파일의 연산

- 운영체제는 파일에서 다양한 연산 지원하여 컴퓨터에서 파일을 사용하도록 함
  - 파일 생성하기
  - 파일 열기
  - 파일 쓰기
  - 파일 읽기
  - 파일 재설정
  - 파일 삭제
  - 파일 크기 조절
  - 속성 설정
  - 파일 이름 바꾸기
  - 파일 삭제

10. 파일 디스크립터

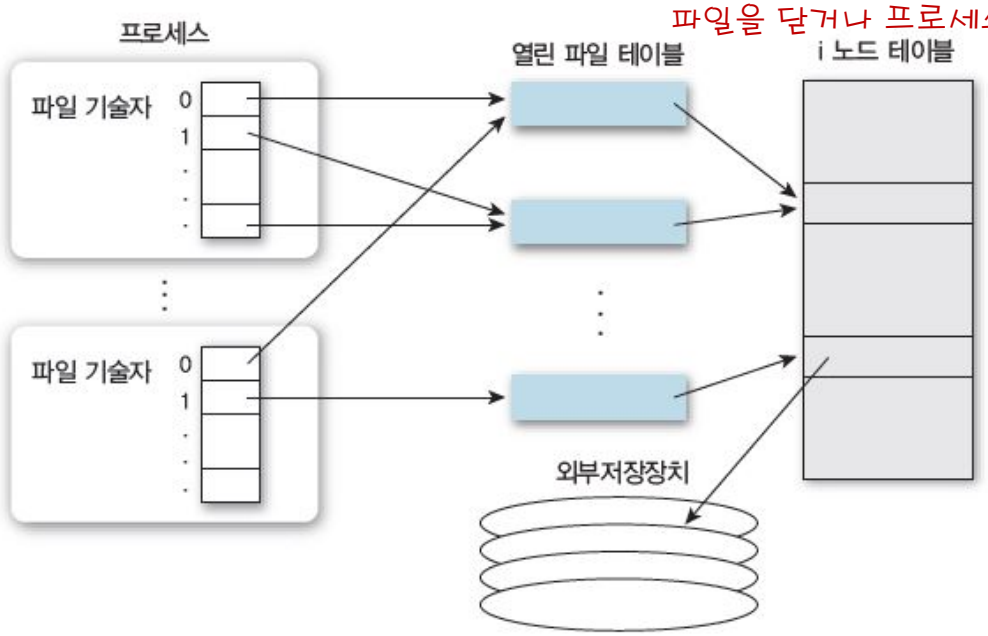
■ 파일 디스크립터descriptor(기술자)

- 파일을 액세스하는 동안 운영체제에 필요한 정보를 모아 놓은 자료구조

파일마다 독립적으로 존재
(b)와 같이 파일을 열 때 프로세스가 생성
파일 디스크립터는 음이 아닌 고유의 정수로, 파일을 액세스하려고 열린 파일(테이블)을 식별하는 데 사용
시스템에 따라 구조가 다를 수 있으며, 파일 시스템이 관리하여 사용자 직접 참조 불가
디스크의 모든 파일이나 디렉터리는 디스크에 저장하므로, 열린 각 파일이나 디렉터리의 파일 디스크립터는 디스크에 저장했다가 파일을 열면 메모리에 복사
파일을 닫거나 프로세스 종료할 때 폐기

- 파일 이름, 크기, ID(번호), 구조, 저장 주소(디스크 내)
- 공유 가능
- 액세스 제어 정보
- 생성 날짜(시간)
- 저장장치 정보

(a) 파일 디스크립터의 내용



(b) 파일 디스크립터의 역할

그림 10-8 파일 디스크립터

11. 파일에 엑세스 하는 방법

■ 순차 액세스

- 파일에 있는 정보는 레코드 단위의 순서로 처리하는 것이 가장 일반. 파일에서 대부분의 동작은 읽기와 쓰기. 순차 액세스에서 읽기 동작은 파일의 다음 부분을 읽은 후 자동으로 파일 포인터 증가시킴
- 쓰기 동작은 파일의 끝에 내용 추가하고, 포인터를 쓴 내용(파일의 새로운 끝)의 끝으로 이동. 순차 파일은 일반적으로 데이터 입력 파일에는 사용하지 않고 프로그램의 임시 작업 파일로 사용. 이런 파일은 시작 위치로 재설정 가능, 어떤 시스템에서 프로그램은 정수 n 개의 레코드를 앞뒤로 건너 뛰기 가능
- 파일의 테이프 모델 기반
- 개념도

그림 10–9 **순차 액세스**

11. 파일에 엑세스 하는 방법

■ 직접 엑세스

- 모든 블록을 직접 읽거나 쓸 수 있으며, 읽기나 쓰기 순서 없음
- 대규모 데이터베이스에서 유용한 방법, 파일의 디스크 모델 기반
- 순차 액세스의 구현과 직접 액세스의 구현 비교

reset(*초기화*)	`cp = 0;`
read next(*다음 것을 읽음*)	`read cp;` `cp = cp + 1;`
write next(*다음 것을 기록*)	`write cp;` `cp = cp + 1;`

(a) 순차 액세스 (b) 직접 액세스

그림 10–10 **순차 액세스와 직접 액세스의 구현 비교**

11. 파일에 엑세스 하는 방법

인덱스 순차 액세스ISAM, Indexed Sequential Access Method

- 직접 액세스를 기반으로 디스크의 물리적 특성에 따라 인덱스를 구성하여 탐색, 포인터 사용하여 파일 액세스
- 큰 파일도 적은 입출력으로 탐색 가능, 인덱스도 1~2차 인덱스 파일로 구성 처리 가능
- 파일에서 특정 항목을 찾으려면, 먼저 인덱스 탐색, 그런 다음 포인터 사용하여 파일 직접 액세스해서 원하는 항목 찾아냄

11. 파일에 엑세스 하는 방법

■ 인덱스 순차 액세스 : ISAM 파일 예

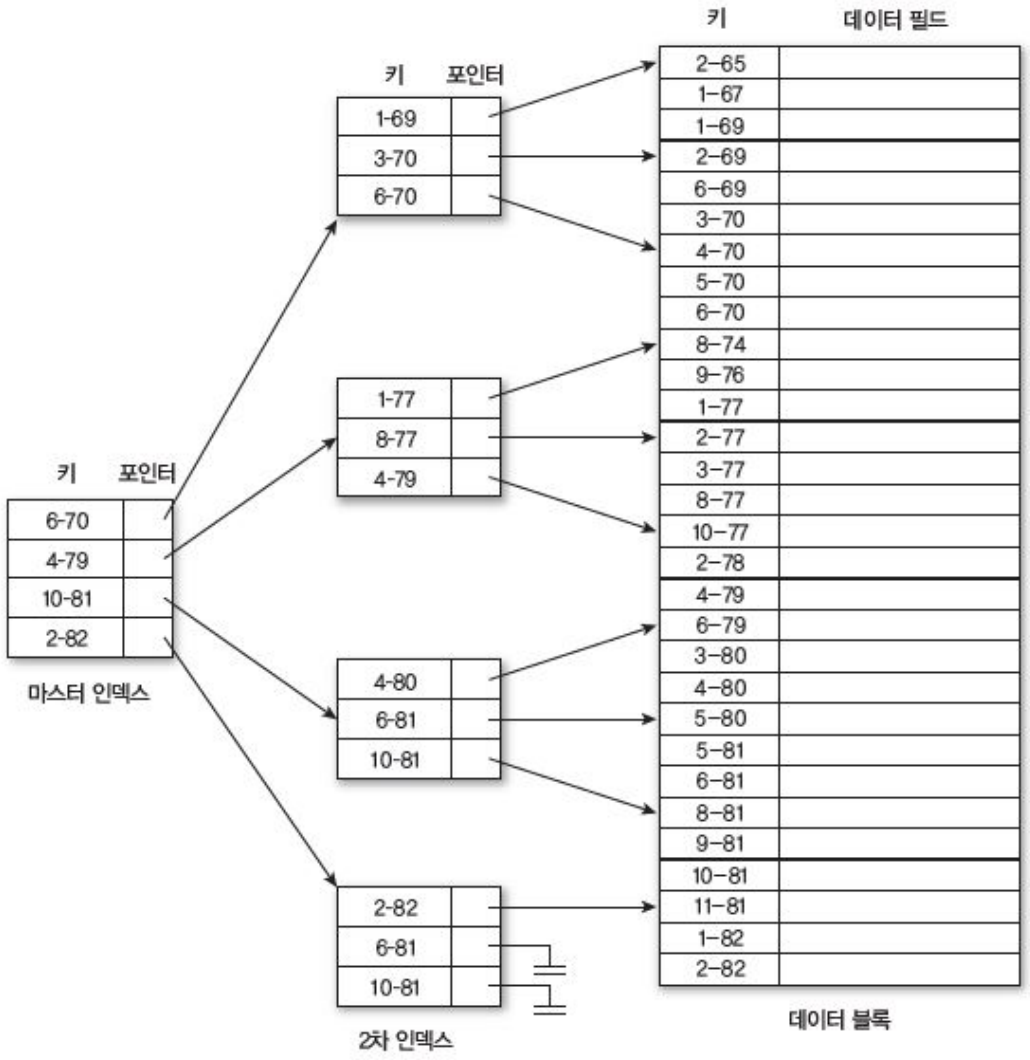


그림 10-11 인덱스 순차 액세스 : ISAM 파일 예

Section 02 파일을 관리하는 디렉터리 시스템 (1. 개념)

디렉터리의 개념

- 디렉터리를 유지하고 관리하여 디스크 등에 저장된 파일을 관리
- 디렉터리의 구분
  - 장치 디렉터리 : 각 실제 장치에 저장. 장치에 있는 파일의 물리적 속성, 파일의 위치, 파일의 크기와 할당 과정 등을 나타냄
  - 파일 디렉터리 : 모든 파일의 논리적 구성으로 이름, 파일 유형, 소유한 사용자, 계정 정보, 보호 액세스 코드 등을 기술
- 파일 시스템에서 다른 파일들의 이름과 위치 정보(파일 인덱스)를 담은 파일로, 다른 파일들과 달리 사용자 데이터 저장하지 않음
- 장치의 범위를 확장할 수 있고 다른 디스크 장치들을 포함 가능
- 디렉터리에 있는 정보 중 일부는 사용자나 응용 프로그램이 이용 가능하나, 대부분 시스템 루틴이 사용자에게 간접적으로 제공

1. 디렉터리의 개념

- 디렉터리와 파일의 관계

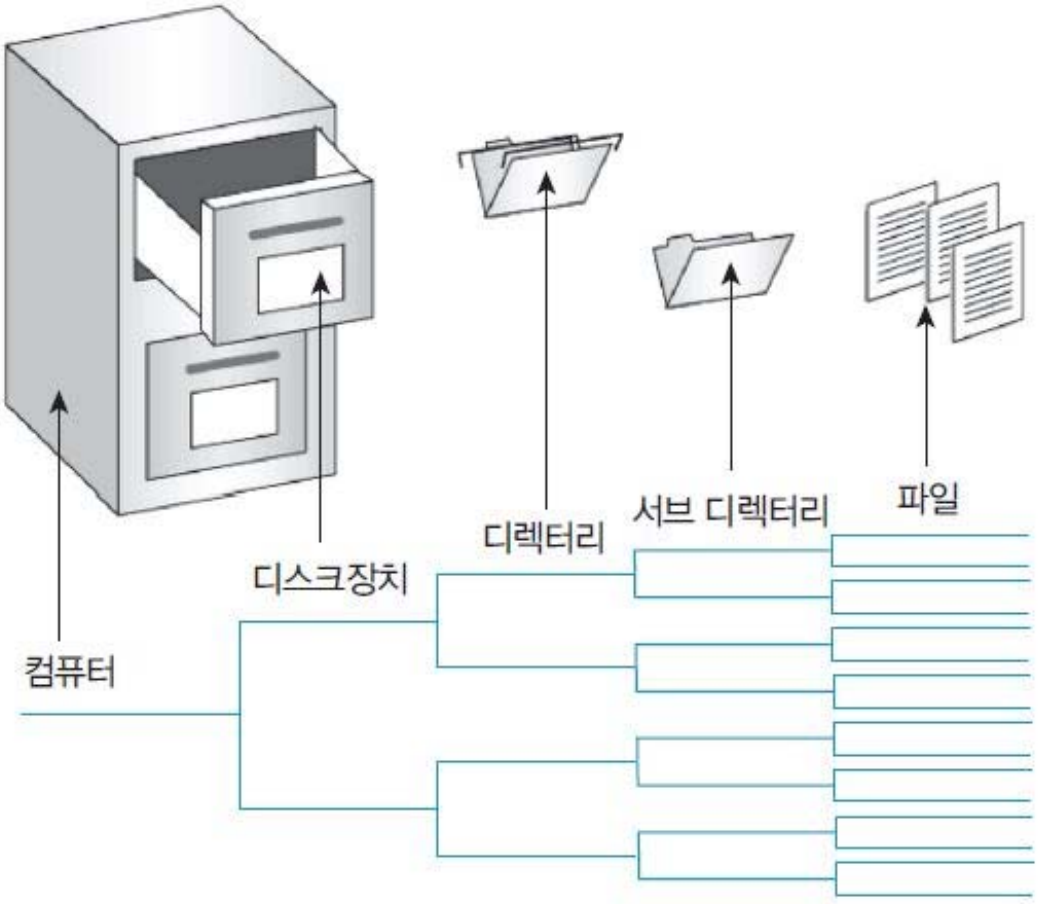


그림 10-12 **디렉터리와 파일**

1. 디렉터리의 개념

- 디렉터리의 파일 정보
  - 파일 이름
  - 파일 형태
  - 위치
  - 크기
  - 현재 위치
  - 보호
  - 사용 수
  - 시간, 날짜, 처리 식별

2. 디렉터리의 구현

■ 디렉터리의 구조

- 디렉터리 공간 할당, 관리하는 방법은 파일 시스템의 효율성과 신뢰성에 큰 영향
- 파일 이름, 파일 인덱스의 내용을 포함하는 파일로 구현
- 계층적으로 구성

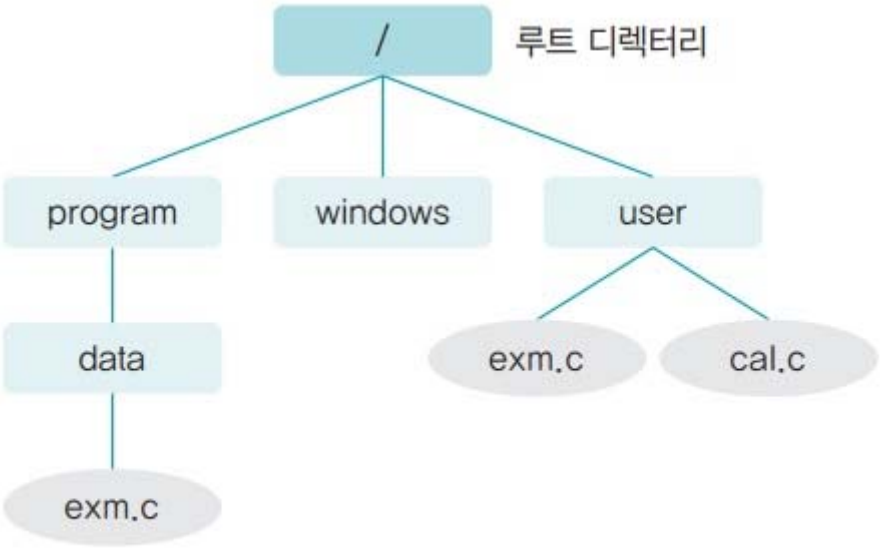


그림 11-14 디렉터리 계층 구조의 예

2. 디렉터리의 구현

- '/수학과/교수/programs/a3/p3.c' 경로명의 디렉터리와 속성으로 디렉터리 구현 예

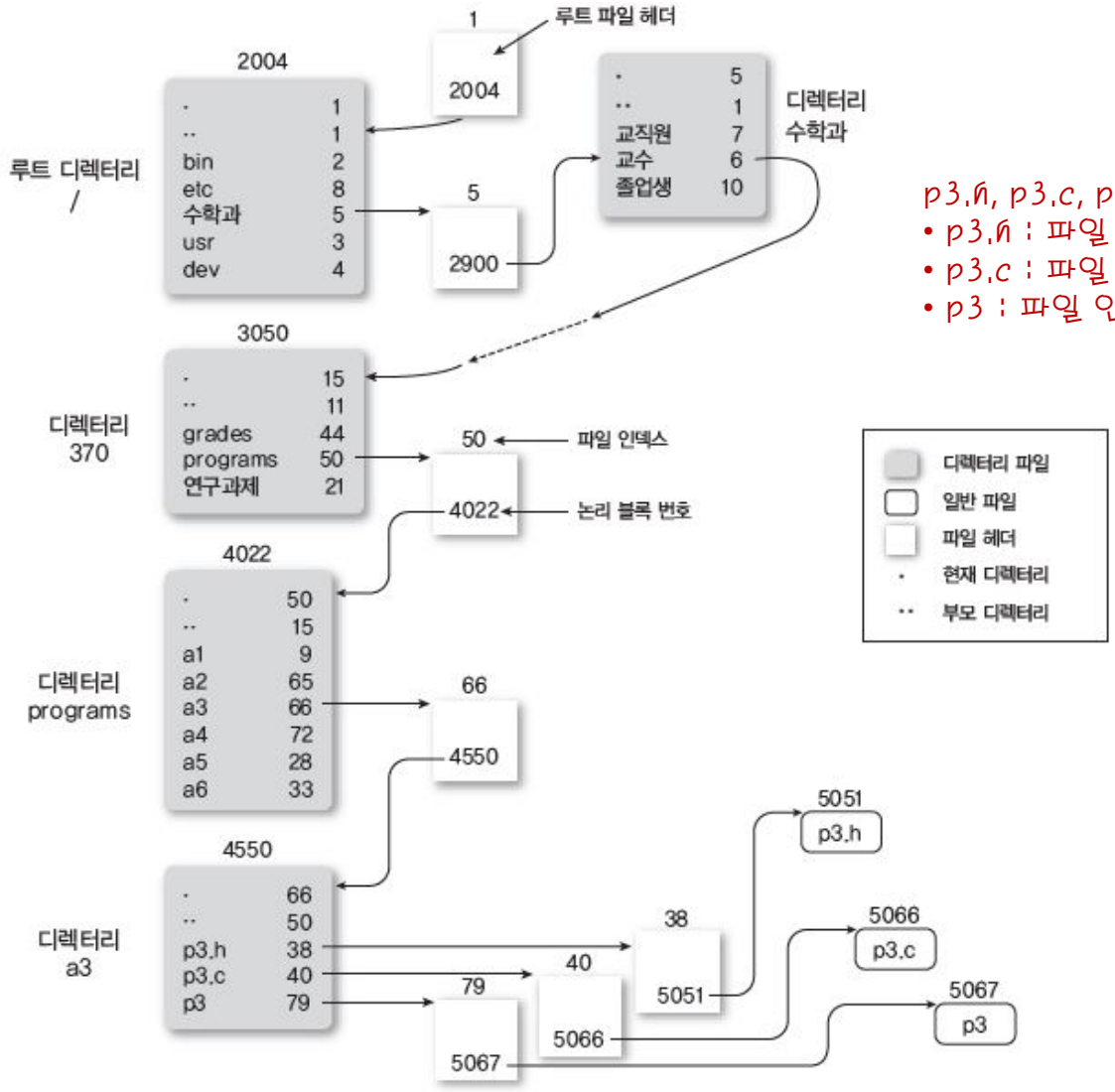


그림 10-13 디렉터리의 구조

p3.h, p3.c, p3 의 파일 헤더,
- p3.h : 파일 인덱스 38, 논리 블록 번호 5051
- p3.c : 파일 인덱스 40, 논리 블록 번호 5066
- p3 : 파일 인덱스 79, 논리 블록 번호 5067

2. 디렉터리의 구현

선형 리스트를 이용한 디렉터리 구현

- 디렉터리에 파일 이름, 포인터들의 선형적 리스트 구성하여 파일의 생성과 삭제 등을 실행
  - 선형 탐색을 해야 파일을 찾을 수 있어 오버헤드가 증가하므로 시스템 성능 떨어뜨릴 수 있는 단점
  - 사용된 디렉터리 정보를 저장하는 소프트웨어 캐시를 구현하여 정보를 매번 디스크에서 읽어 오지 않도록 하여 문제 해결(캐시적중).
- 리스트 정렬하여 이진 탐색 방법 사용
  - 평균 탐색 시간은 줄일 수 있으나, 리스트가 정렬 상태를 유지하려고 하면 파일 생성과 삭제 복잡
  - 이진 연결 트리 사용하여 문제 해결

2. 디렉터리의 구현

■ 해시 테이블hash table을 이용한 디렉터리 구현

- 해시 테이블을 이용하여 파일 이름을 제시하면, 해시에서 값을 얻어 리스트를 직접 액세스하도록 디렉터리 구현 가능
  - 디렉터리 탐색 시간을 줄이면서 성능을 개선하기에 많이 사용
  - 둘 이상의 파일 이름이 같은 위치를 지정할 때는 충돌 발생 가능성, 보완하면 쉽게 삽입·삭제 가능
  - 해시 테이블의 크기가 고정되고 크기에 따라 해시 기능이 제한을 받는다는 문제점
  - 해시 테이블의 크기는 파일의 크기 정보 활용
  - 해시 테이블에서 충돌을 해결하는 방법으로 체인 오버플로 해시 테이블 시용
    - 각 해시 항목은 하나의 값이 아니라 연결 리스트가 되고, 새로운 항목을 연결 리스트에 추가하여 충돌 해결
    - 이름을 찾으려면 연결 리스트를 살펴봐야 하기 때문에 찾기 작업은 다소 늦어지나, 전체 디렉터리를 선형으로 찾는 것보다는 훨씬 빠름

3. 디렉터리의 연산

디렉터리의 연산

- 탐색
- 파일 생성
- 파일 삭제
- 파일 열람
- 파일 이름 변경
- 파일 시스템 순회
- 백업

4. 디렉터리의 구조

1단계single level 디렉터리

- 가장 간단한 구조(장치 디렉터리)
- 모든 파일이 동일한 디렉터리에 있어 유지하고 이해 용이
- 파일 수가 증가하거나 다수의 사용자가 있을 때 모든 파일이 동일한 디렉터리에 있으므로 모두 고유한 이름을 가져야 하는 불편
- 파일 이름은 보통 내용과 관련되어 있지만, 시스템이 정하는 길이의 제한 받음
- 사용자가 한 사람이라도 파일 수가 많을 때는 고유한 이름으로 새 파일을 생성해야 하므로 모든 파일 이름 기억해야 함
- 구조

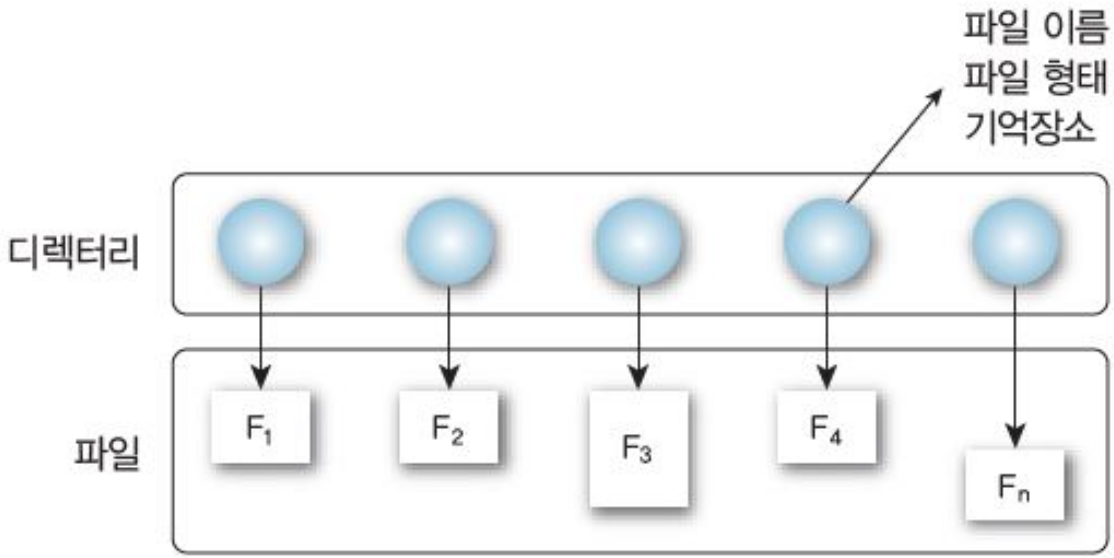


그림 10-14 1단계 디렉터리

4. 디렉터리의 구조

2단계 디렉터리

- 사용자들이 자신의 서브디렉터리 생성, 그곳에 자신의 파일을 구성하는 것(유닉스, 도스)
- 사용자 간 파일 이름이 섞이지 않도록 각 사용자는 다른 디렉터리 만들어 사용
- 모든 파일이 물리적으로 동일 장치에 있어 대형 시스템에서는 사용자 디렉터리를 논리적으로 구성
- 루트는 마스터 파일 디렉터리이고, 아래로 사용자 파일 디렉터리와 그 아래로 파일이 있음
- 파일은 트리의 리프에 해당. 사용자 이름과 파일 이름은 루트 디렉터리에서 리프까지 경로로 정의하는데, 이를 경로명이라고 함
- 시스템 내의 모든 파일의 경로명은 유일하며, 원하는 파일의 경로명을 알아야 해당 파일 지정 가능
- 구조

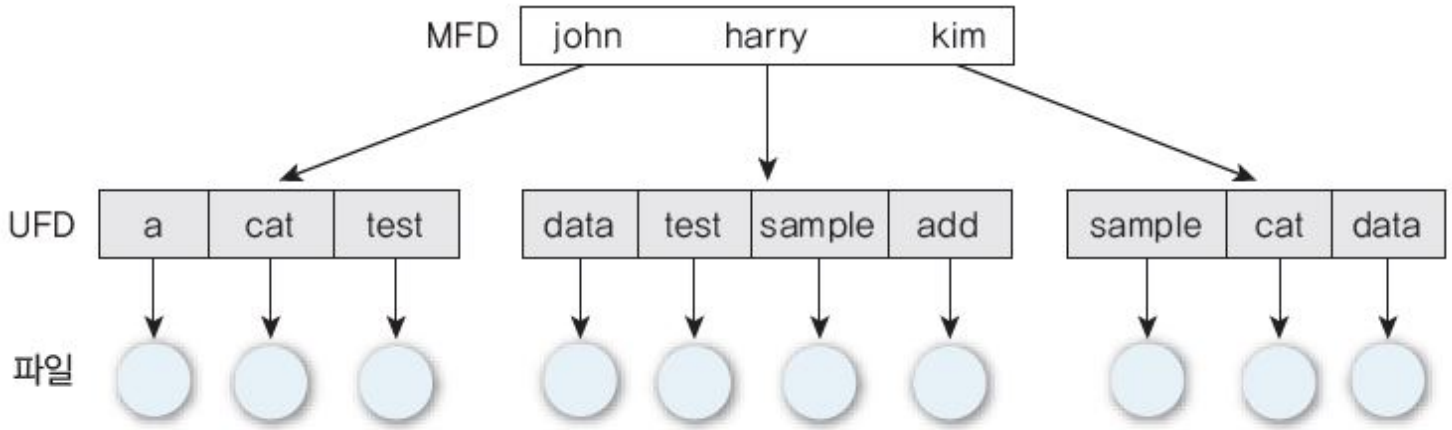


그림 10-15 2단계 디렉터리 구조

4. 디렉터리의 구조

- 각 사용자는 자신의 사용자 파일 디렉터리UFD, User File Directory를 가짐
  - 사용자 파일 디렉터리는 비슷한 선형, 이진, 해시 등 구조로 오직 한 사용자의 파일만 나타냄
  - 파일 이름이 충돌하는 문제를 해결하고 다른 사용자가 액세스할 수 없다는 장점
  - 반면에 두 사용자가 한 파일을 공유해서 사용할 때는 문제가 발생
- 2단계 디렉터리 구조에서 한 사용자의 업무를 시작하거나 새로 등록(로그인)할 때는 각 사용자 이름이나 계정 번호 인덱스, 사용자 디렉터리에서 각 항목 포인터가 있는 마스터 파일 디렉터리MFD, Master File Directory 먼저 탐색
- 사용자가 특정 파일을 참조할 때 자신의 디렉터리만 탐색하므로 각 사용자 파일 디렉터리에 있는 모든 파일의 이름이 고유해도 사용자 사이에서는 동일한 파일 이름 있을 수 있음
- 파일을 삭제할 때도 운영체제는 지역 UFD만 탐색하므로 이름이 동일한 다른 사용자 파일 삭제하지 않음
- 사용자 디렉터리는 필요에 따라 생성·삭제 가능.
- 특수한 시스템 프로그램에서는 해당 사용자 이름과 계정한 정보를 미리 준비해 두었다 한 사용자가 파일 디렉터리를 생성할 때 해당 파일 항목을 마스터 파일 디렉터리에 추가하기도 함

4. 디렉터리의 구조

트리 구조 디렉터리

- 트리 구조 디렉터리 중 높이가 2인 트리가 2단계 디렉터리
- 트리는 루트 디렉터리가 하나 있고, 시스템 내의 모든 파일의 경로명은 유일
- 각 디렉터리에는 서브디렉터리나 파일이 있으며, 모든 디렉터리는 내부적으로 형식 동일
- 보통 각 사용자에게는 현재 디렉터리가 있다. 파일 참조가 일어나면 현재 디렉터리 검색
- 현재 디렉터리에 없는 파일 사용하려면 경로명 직접 입력하거나 현재 디렉터리 바꿔야 함
- 디렉터리는 시스템 호출로 변경 가능.
- 사용자가 작업을 시작하거나 사용자가 등록할 때 운영체제 이 사용자 항목 탐색
- 첫 번째 사용자 디렉터리 포인터는 계정 파일에 들어 있고, 사용자의 지역변수를 정의하여 사용자의 현재 디렉터리를 명시
- 경로명에는 절대 경로와 상대 경로가 있음
- 빈 디렉터리라면 삭제가 간단. 그러나 비어 있지 않은 디렉터리를 삭제하려면, 먼저 해당 디렉터리에 있는 모든 파일을 제거해야 함

4. 디렉터리의 구조

- 구조

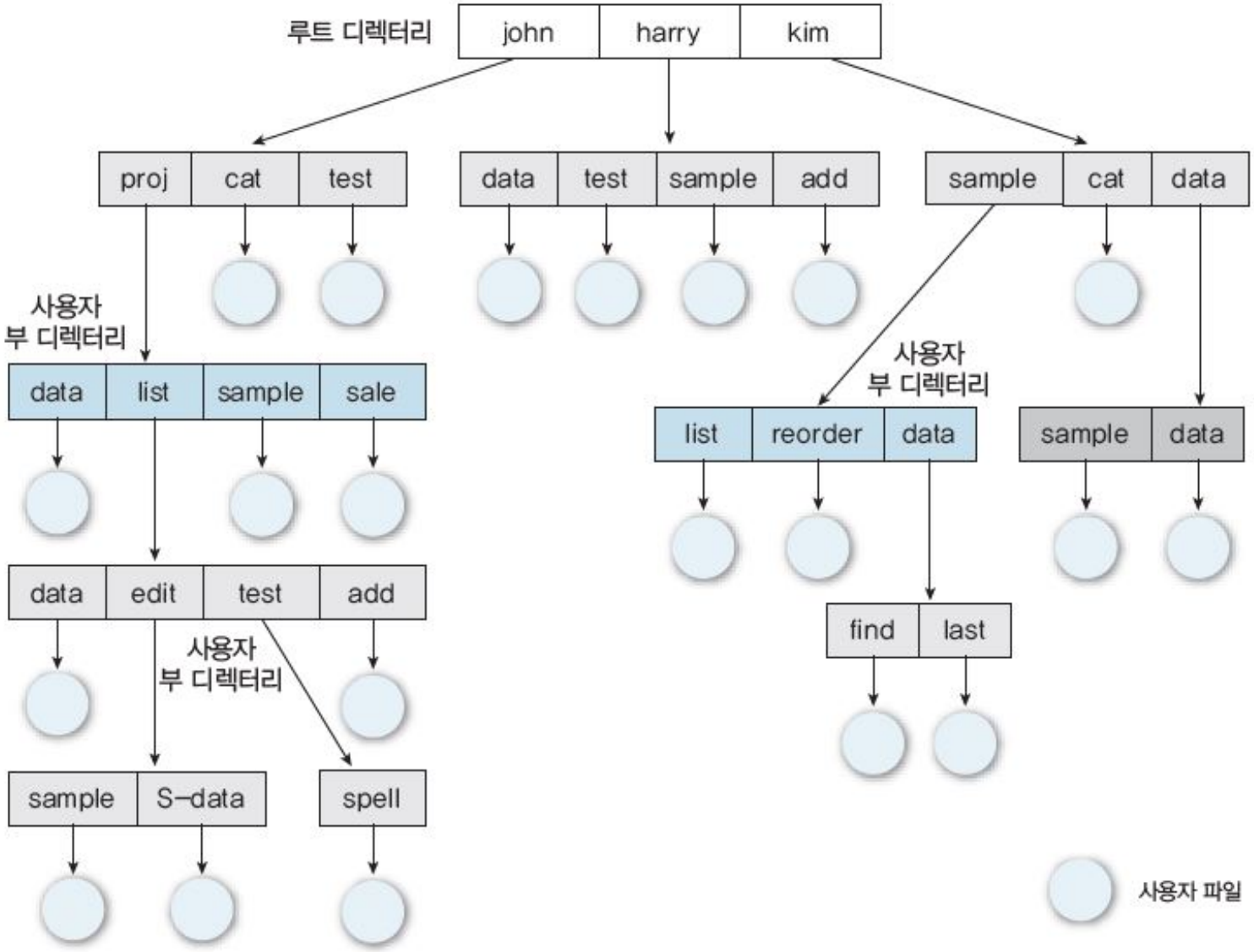


그림 10-16 트리 구조 디렉터리

4. 디렉터리의 구조

비순환 그래프acyclic graph 디렉터리

- 트리 구조 디렉터리를 확장하여 일반화
- 트리 구조 디렉터리에서는 파일이나 디렉터리의 공유 금지한 반면, 이것은 허용
- 사용자가 자신의 디렉터리를 정의하도록 파일이 별도의 구조 가질 수 있게 함
- 동일한 파일이나 서브디렉터리는 다른 디렉터리에 있을 수 있음
- 공유는 복사의 개념이 아니고, 공유 파일은 파일 복사본과 다름
- 공유 파일과 서브디렉터리를 구현하는 일반적인 방법은 링크 이용
- 트리 구조 디렉터리보다 융통성은 좋으나 복잡

4. 디렉터리의 구조

- 구조

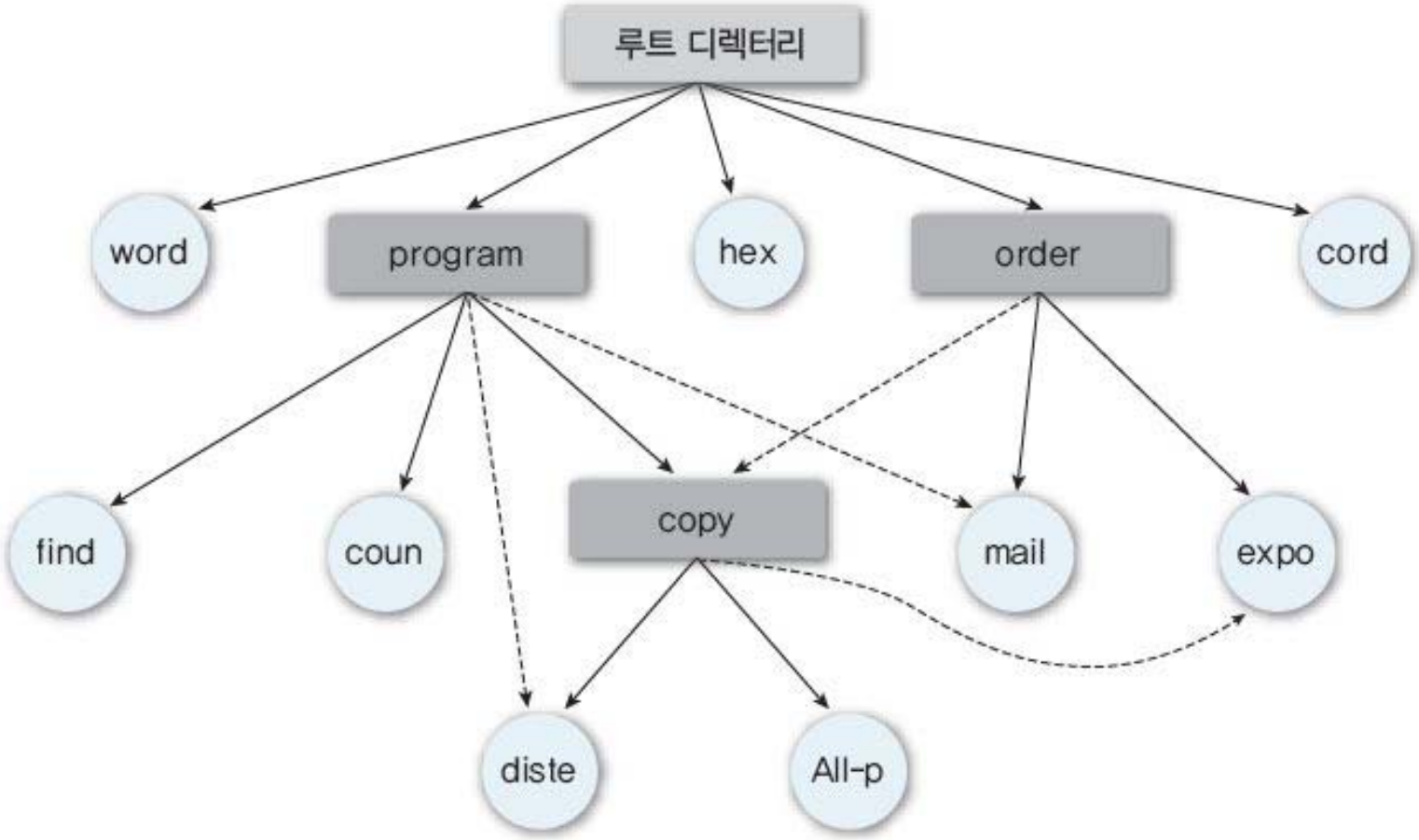


그림 10-17 **비순환 그래프 디렉터리**

4. 디렉터리의 구조

■ 일반 그래프 디렉터리

- 2단계 디렉터리에서 사용자에게 서브디렉터리를 생성하도록 하면 트리 구조 디렉터리가 됨
- 트리 구조 디렉터리에 새로운 파일을 첨가하거나 서브디렉터리 첨가하면 트리 구조는 유지하지만, 링크를 가지면 트리 구조가 파괴되어 일반 그래프 디렉터리 됨
- 구조

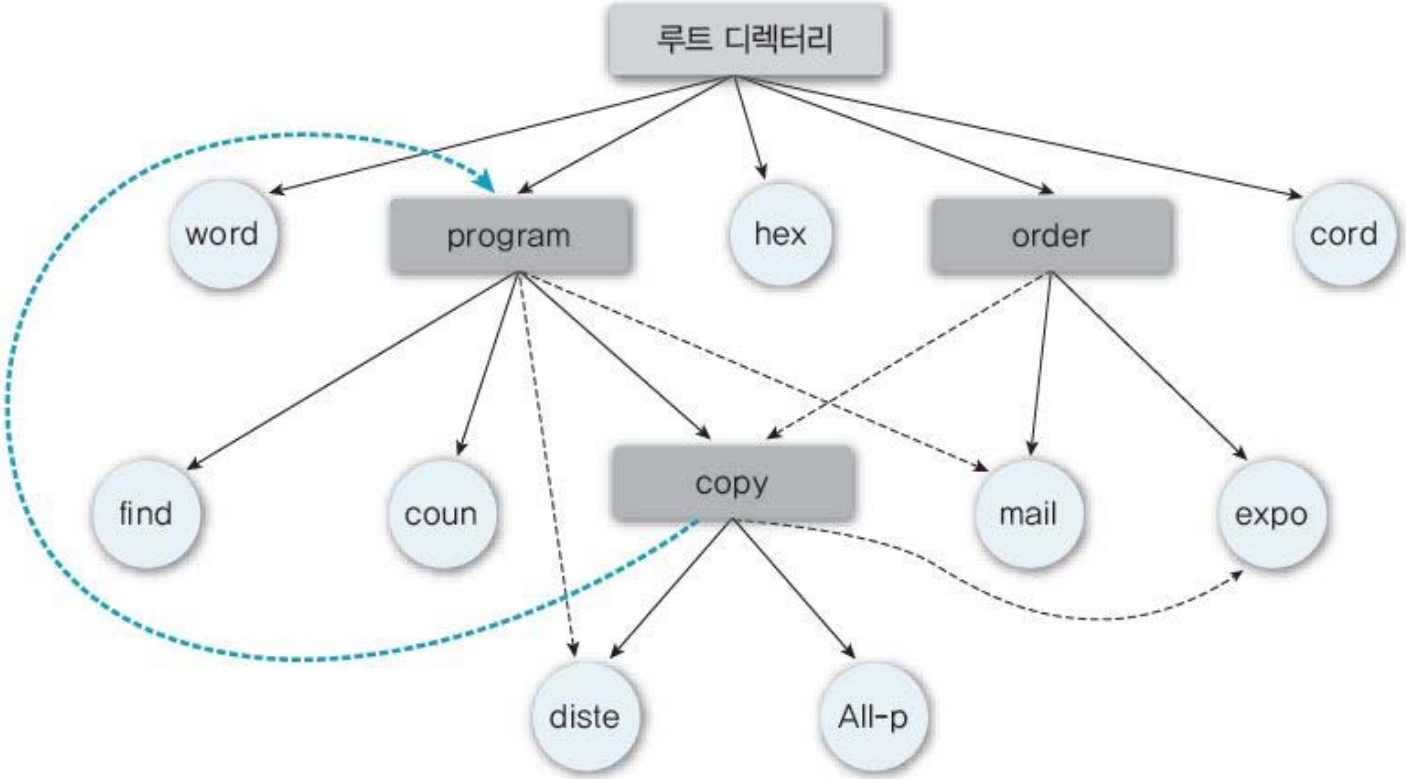


그림 10-18 그래프 디렉터리

Section 03 파일의 디스크 할당(파일의 디스크 할당 방법)

연속 할당

- 파일을 디스크의 연속적인 주소에 할당하는 방법
- 메인 메모리의 동적 분할 방법과 비슷
- 순서가 있어 블록 b 다음에 블록 b+1을 액세스하는 데 헤드 이동 필요 없음
- 헤드가 한 실린더의 마지막 섹터에서 다음 실린더의 첫 번째 섹터만 1트랩 이동시키면 됨
- 연속적으로 할당된 파일들에 액세스하는 데 필요한 디스크 탐색 횟수 최소화
- 한 파일의 연속 할당은 주소와 첫 번째 블록 수로 정의
- 순차 액세스와 직접 액세스 모두 연속 할당 방법 지원

1. 파일의 디스크 할당 방법

- 연속 할당의 개념도

디렉터리

파일	시작 블록	길이
CTR.doc	1	4
SaleRep	11	7
Test	21	3

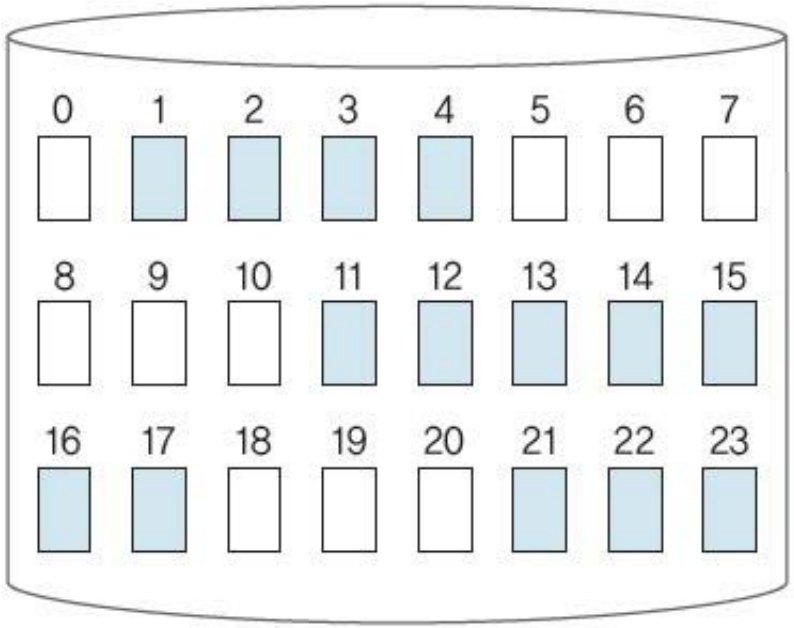


그림 10-19 **디스크 공간의 연속 할당**

1. 파일의 디스크 할당 방법

- 연속 할당의 문제점
  - 외부 단편화
    - 파일을 할당하고 삭제하면서 디스크 공간을 작은 조각들로 나눔. 전체 디스크 공간이 요구량을 만족할 때도 연속된 공간이 아니면 외부 단편화 발생.
    - 디스크 전체 크기와 평균 파일의 크기에 따라 다름.
    - 사용자가 전체 파일 시스템을 다른 플로피 디스크나 테이프에 복사하는 재포장 루틴을 수행한 후 플로피 디스크 완전히 비워서 커다란 연속 사용 가능 공간 생성하면 해결 가능
  - 파일 공간 크기를 결정하는 어려움
    - 파일에 공간이 얼마큼 필요할지 결정하기가 곤란.
    - 해결 방법
      - » 사용자 프로그램이 오류 메시지를 출력하고 종료하면, 사용자가 더 많은 공간을 할당하여 프로그램을 다시 수행
      - » 아주 큰 공간을 찾아 파일의 항목을 새로운 공간에 복사하고 이전 공간은 해제하는 방법

1. 파일의 디스크 할당 방법

■ 연결 할당

- 각 파일을 디스크 블록들의 리스트에 연결, 디스크 블록들은 디스크내에 흩어져 있어도 되는 방법
- 디렉터리는 파일의 첫 번째 블록 포인터를 가지고 있음
- 블록 구조

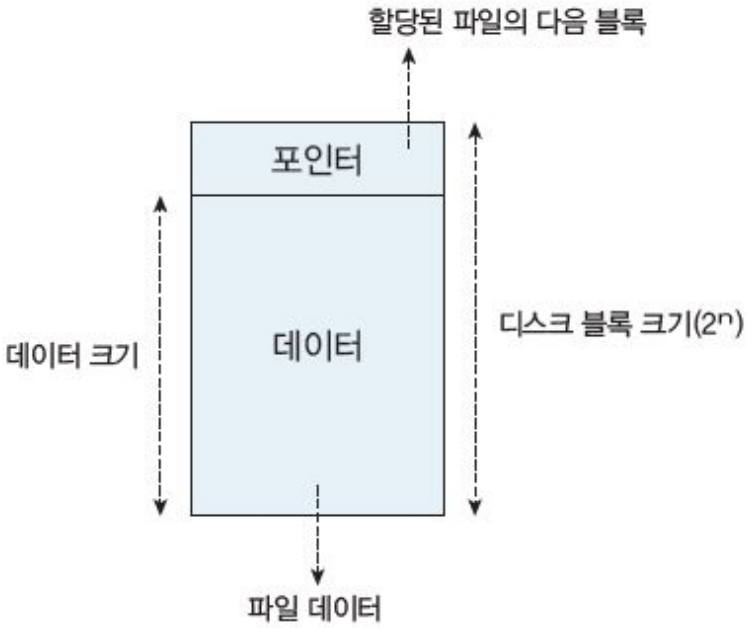


그림 10-20 **블록 구조**

1. 파일의 디스크 할당 방법

- 디스크 공간의 연결 할당

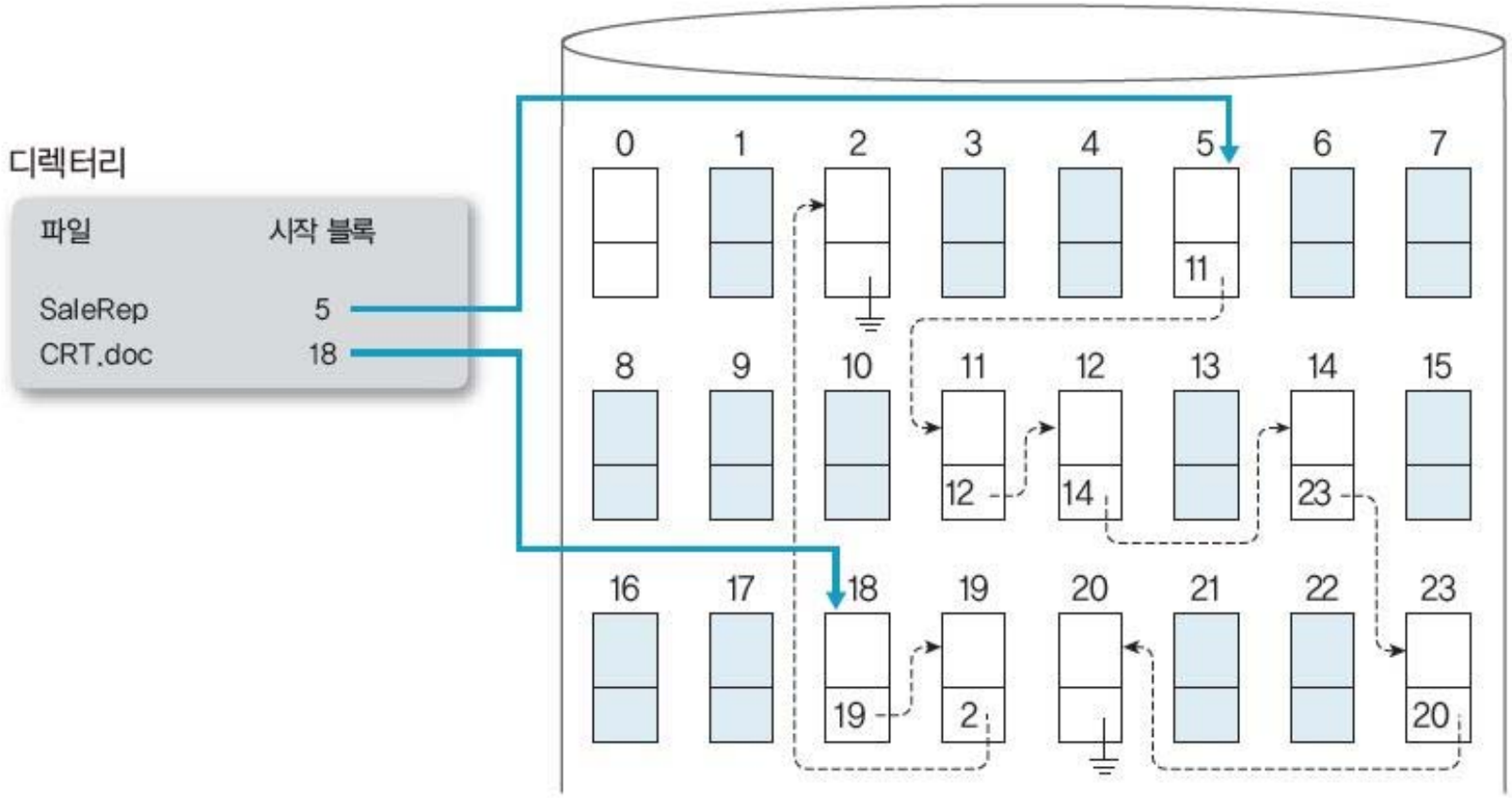


그림 10-21 디스크 공간의 연결 할당

1. 파일의 디스크 할당 방법

- 파일을 생성 용이
- 파일을 읽을 때는 블록에서 블록까지 포인터를 따라가면서 읽음
- 파일을 쉽게 확장할 수 있으며, 블록당 디스크 포인터 공간이 필요함에도 쉽게 구현
- 외부 단편화가 없음
- 블록이 크면 내부 단편화가 발생할 수 있으나, 파일 데이터에 액세스하는 데 필요한 입출력 연산의 횟수는 감소
- 블록이 작으면 데이터를 여러 블록에 분산시켜 성능이 떨어질 수 있음
- 모든 블록이 함께 연결되기 때문에 빈 공간 리스트에 사용 가능한 각 블록은 요구 사항에 맞게 이용 가능
- 문제점
  - 순차 액세스 파일에만 적합
  - 포인터 공간 필요
  - 신뢰성 유지 어려움
  - 탐색 시간 증가
- 이중 연결 리스트 사용하거나 각 블록 내의 관련 블록과 파일 이름 저장하여 부분적 해결

1. 파일의 디스크 할당 방법

- 연결 할당의 중요한 변형은 파일 할당표FAT 사용하는 것
  - 파일 할당표
    - 사용자가 해당 블록의 포인트를 실수로 지우지 않게 하며, 블록 접근을 빠르게 하려고 포인터를 모아 놓은 곳
    - 각 디스크 블록 내에 항목이 한 개 있고, 블록 번호로 참조하며, 연결 리스트로 많이 사용단순하지만 디스크 공간 할당에서는 효율적인 방법으로, 도스와 OS/2 운영체제에 사용
- 디렉터리는 파일의 첫 번째 블록 번호 포함. 그 블록 번호로 인덱스된 테이블 번호는 파일 내 다음 블록의 블록 번호를 가짐
- 테이블 항목 내에서 특별히 끝EOF, End of File 값이 있는 마지막 블록까지 연결을 계속하며, 사용하지 않는 블록들은 제로 테이블 값으로 표시

1. 파일의 디스크 할당 방법

- 파일 할당표

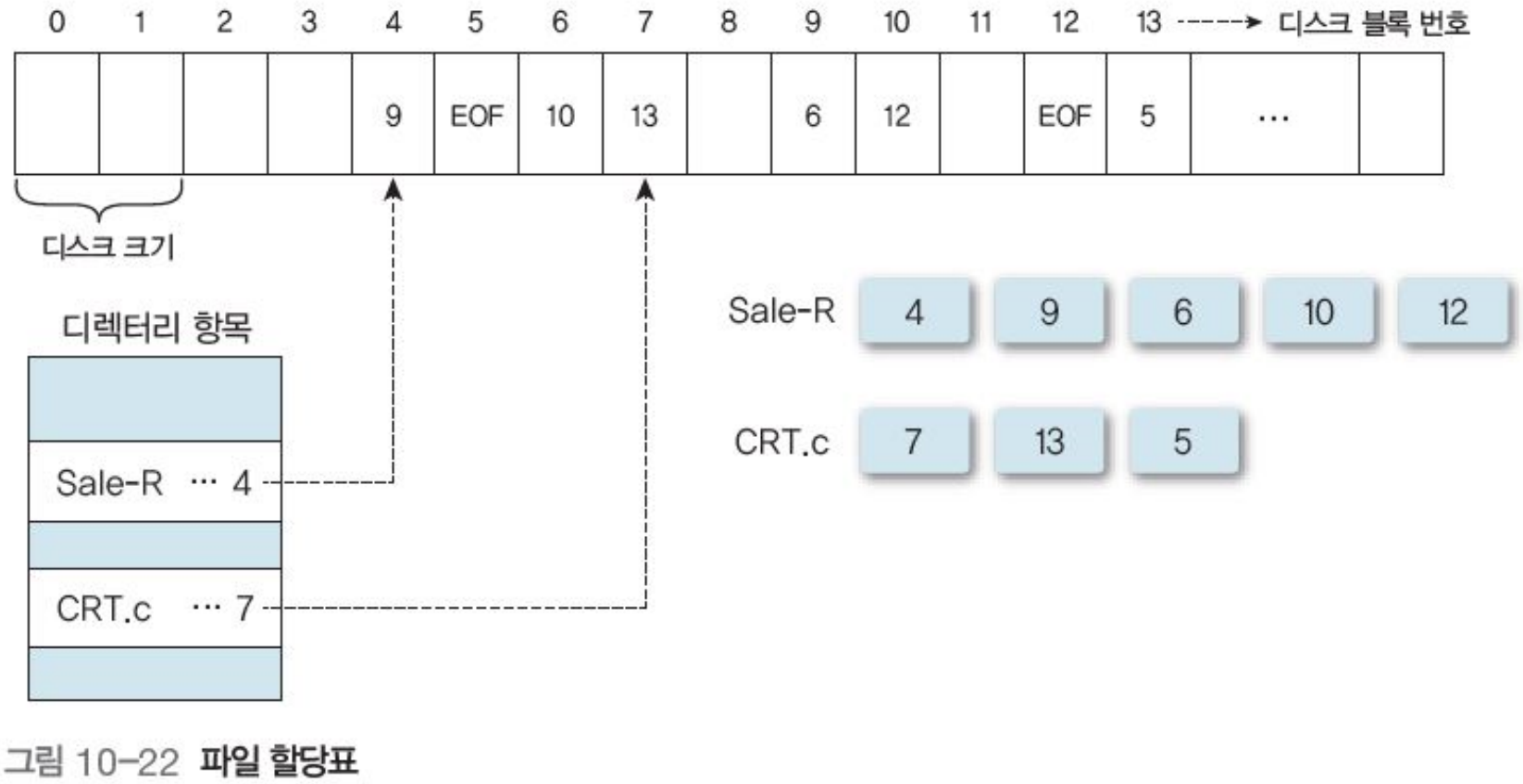


그림 10-22 **파일 할당표**

1. 파일의 디스크 할당 방법

■ 인덱스 할당

- 연속 할당의 외부 단편화와 크기 문제를 해결했으나 직접 액세스를 지원할 수 없고, 블록의 포인터가 디스크 전반에 흩어져 있음
- 모든 포인터를 인덱스 블록이라는 하나의 장소에서 관리하여 직접 액세스 지원
- 각 파일에는 디스크 블록 주소의 배열인 자신만의 인덱스 블록 있음
- 페이징 방법과 비슷. 순차 액세스는 효율성이 떨어지나, 직접 액세스 빠름
- 파일을 생성할 때 인덱스 블록의 모든 포인터는 null 값으로 설정

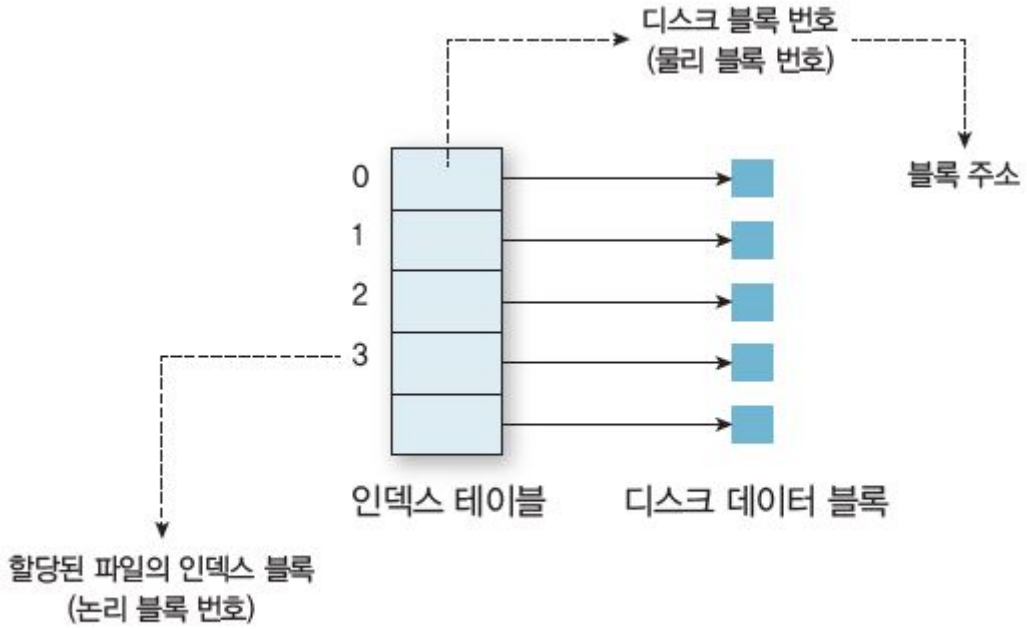


그림 10-23 **인덱스 할당**

1. 파일의 디스크 할당 방법

- 대부분의 파일 크기가 작음
- 전체 인덱스 블록을 할당해야 하며, 포인터 한 개나 2개가 nil이 아니어야 하므로 인덱스 블록이 커짐
- 각 파일은 인덱스 블록을 하나만 가져야 하므로 인덱스 블록의 크기는 가능한 작은 것이 좋으나 인덱스 블록이 너무 작으면 큰 파일들에서는 포인터 충분하지 않음

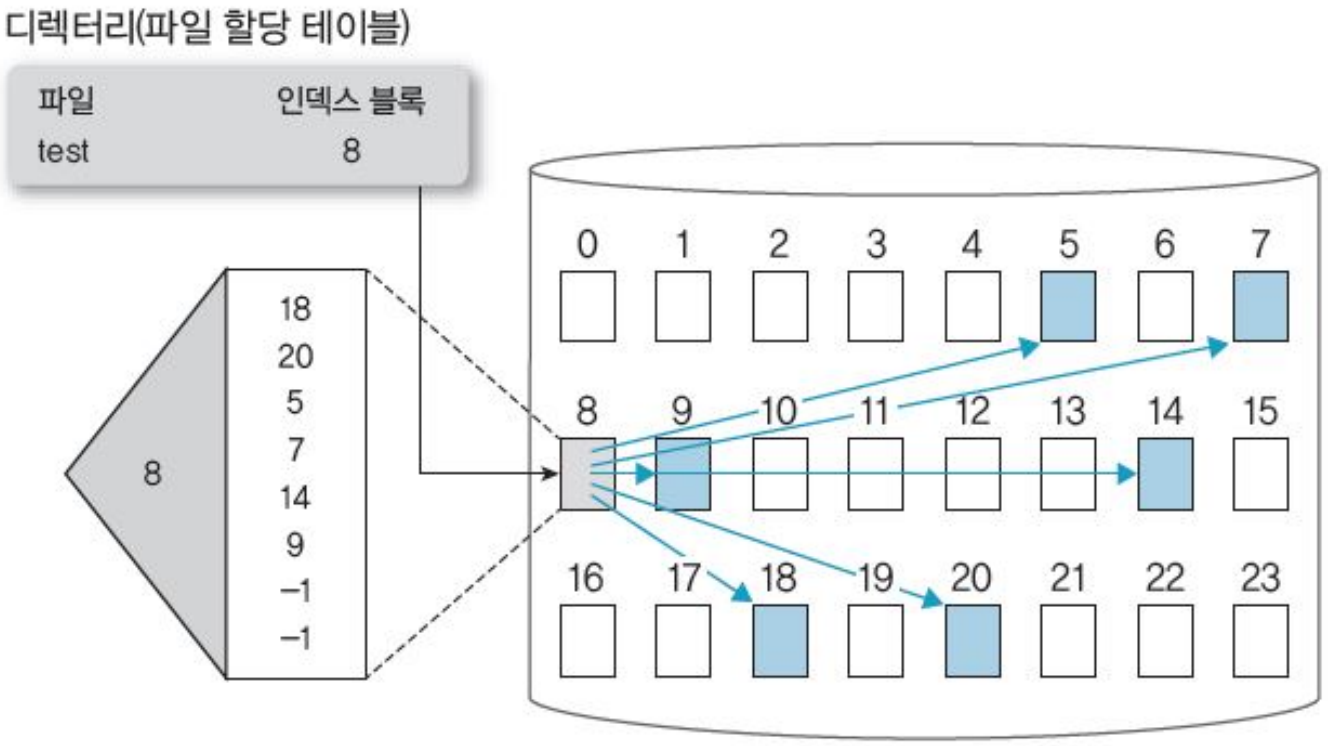


그림 10-24 **디스크 공간의 인덱스 할당**

1. 파일의 디스크 할당 방법

- 구현 쉽고 외부 단편화 없음
- 파일은 블록 배열의 최대 크기로 쉽게 확장할 수 있으나, 블록 배열의 최대 크기로 제한
- 인덱스로 빠르게 임의 액세스(두 번 탐색으로)가 가능하다는 장점
- 순차 액세스는 느릴 수 있지만, 각 블록당 탐색을 요구하는 연결 할당만큼은 나쁘지 않음
- 파일 크기가 블록 인덱스를 초과할 때가 문제
- 인덱스 블록에 사용되지 않는 nil 항목으로 공간 부담이 큼
- 빠르게 액세스하려면 연속 블록 할당해야 함
- 단일 인덱스 파일 할당은 메모리 관리의 단일 페이징 방법과 비슷
  - 페이지 테이블과 같은 인덱스 블록 인덱스(가상 블록 번호)를 항목 값(논리 블록 번호)으로 변환
  - 인덱스 블록을 구성하는 블록의 인덱스로 주소를 지정할 수 있는 블록 범위를 넓히려고 유닉스에서는 여러 항목에 파일 주소를 사용하는 다중 인덱스 파일 할당Multilevel Indexed Allocation, 즉 i 노드를 이용
  - i 노드 : 처음 항목 12개(일부 시스템은 항목 10개)의 데이터 블록 주소LBN가 포함되어 있는 구조

1. 파일의 디스크 할당 방법

- i 노드 구조

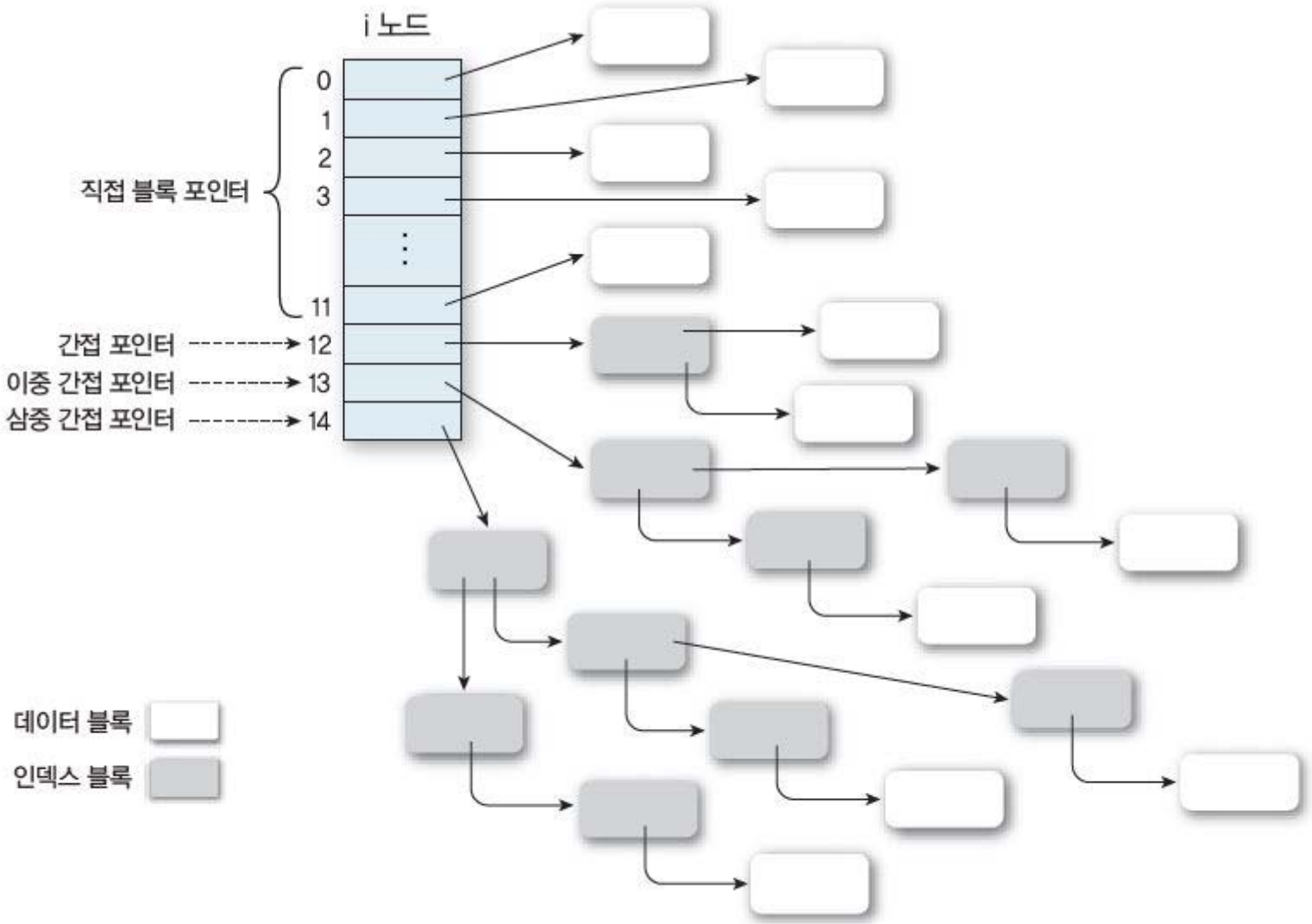


그림 10-25 i 노드 구조

2. 파일의 디스크 할당 방법의 비교

디스크 할당 방법의 비교

- 연속 할당
  - 파일의 직접 액세스를 지원하며, 연결 할당으로 순차 액세스 지원
  - 작은 파일에 효율적, 평균 성능이 아주 좋음
- 연결 할당
  - 메모리에 다음 블록 주소를 보관할 수 있으며, 이를 직접 읽을 수 있음
  - 순차 액세스는 매우 쉬우나, 직접 액세스는 i번째 블록을 읽으려고 디스크를 i번 읽어야 할 때도 있음
- 인덱스 할당
  - 인덱스 블록이 메모리에 있으면 직접 액세스 가능.
  - 메모리에 인덱스 블록을 보관하려면 메모리가 많이 필요
  - 2단계 인덱스는 인덱스 블록을 두 번 읽어야 함
  - 인덱스 할당의 성능은 인덱스 구조와 파일의 크기, 원하는 블록의 위치가 좌우.
  - 파일이 작으면(블록 3~4개 정도) 연속 할당을 사용하고, 파일이 크면 자동 인덱스 할당으로 변환하는 방법을 사용하기도 함

3. 디스크의 빈 공간 관리 방법

3.1 비트맵

- 빈 공간 리스트는 비트맵 또는 비트 벡터로 구현

• 블록 크기=212바이트
• 디스크 크기=1GB(230바이트)
• 블록 수=230/212 =218

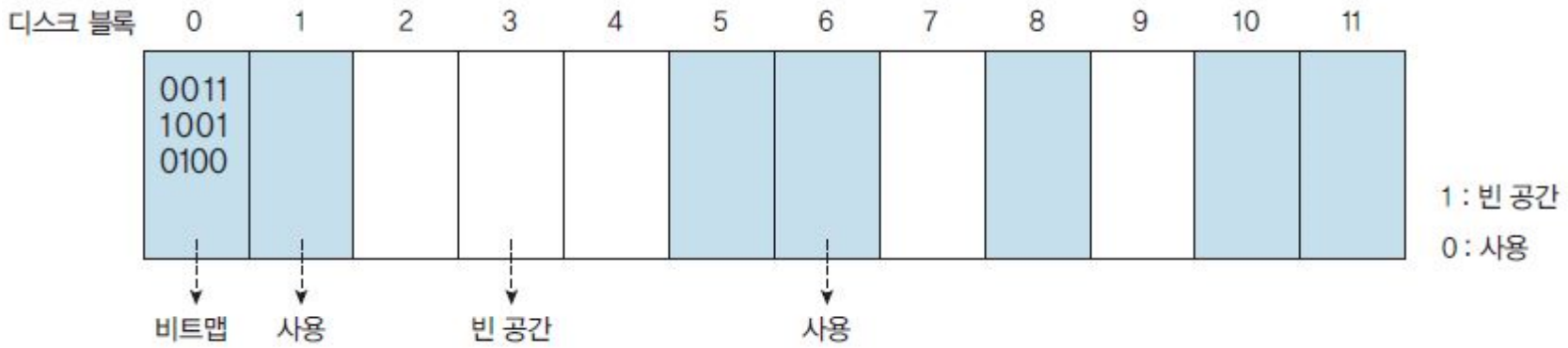


그림 10–26 **비트 벡터**

이 방법은 간편하고 디스크에 연속적인 빈 블록 n개를 찾는 데 효과적인 장점
비트 벡터 전체를 메모리에 보관하지 않으면 비효율적이라서 대형 컴퓨터보다는
마이크로컴퓨터 환경에 더 알맞음

3. 디스크의 빈 공간 관리 방법

- 디스크의 비트맵

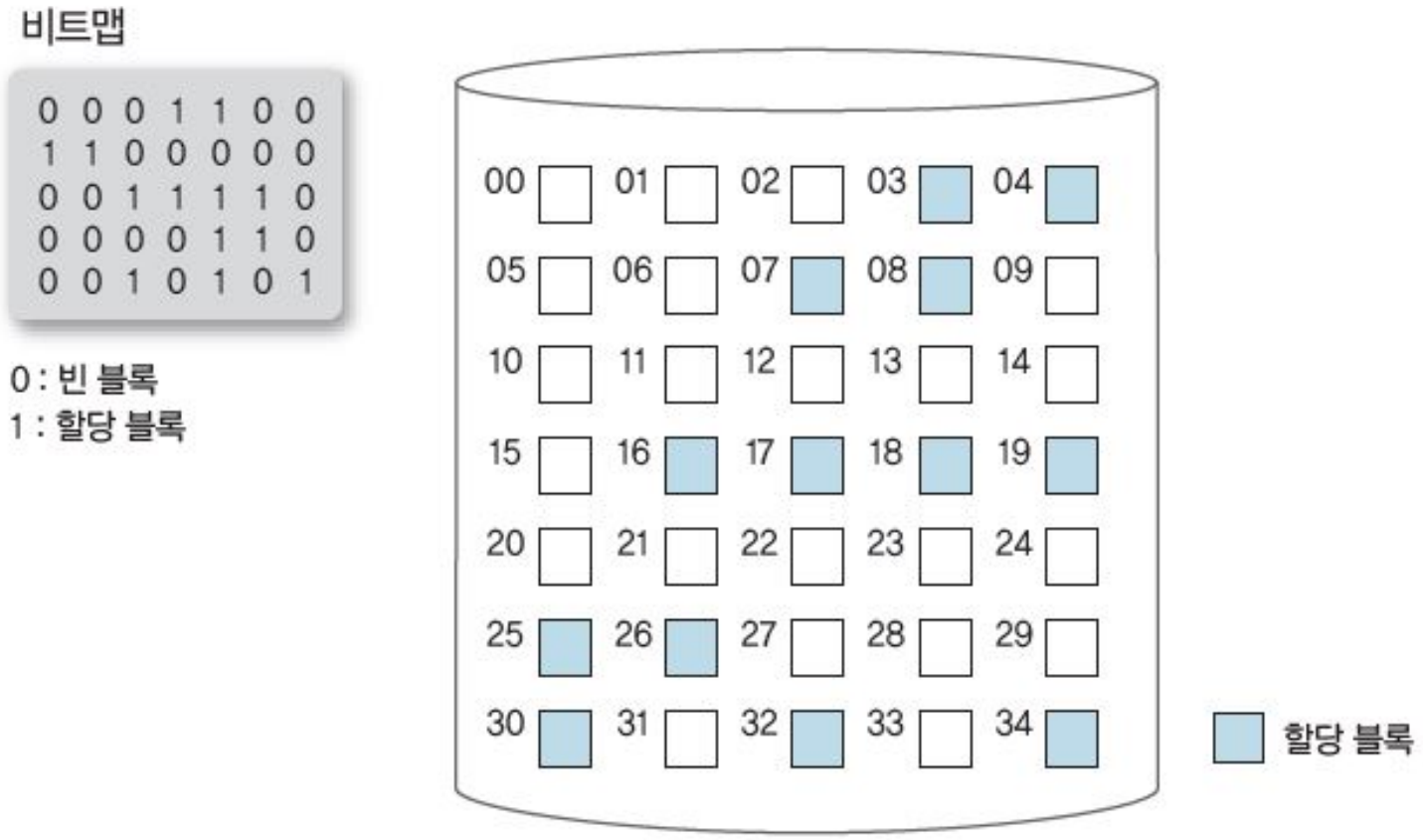


그림 10-27 디스크의 비트맵

3. 디스크의 빈 공간 관리 방법

■ 연결 리스트

▪ 연결 리스트 구현

빈 공간 리스트를 탐색할 때 각 블록을 모두 읽어야 하므로 비효율적

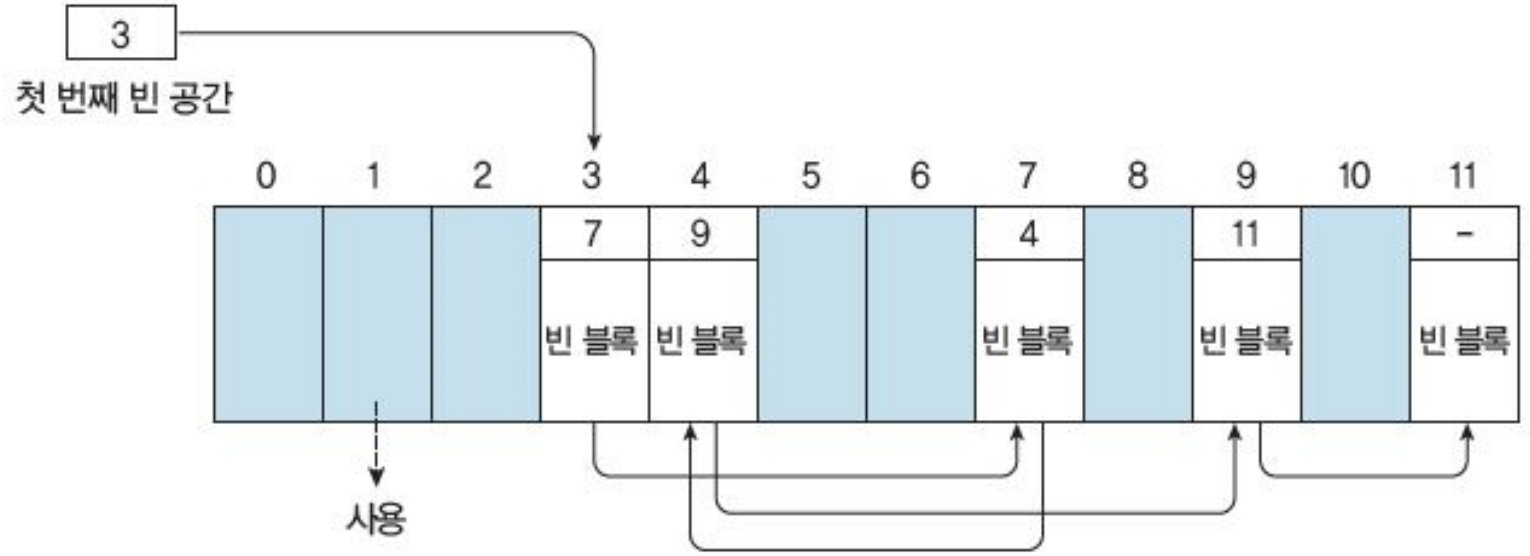


그림 10-28 **연결(링크된) 리스트**

3. 디스크의 빈 공간 관리 방법

- 디스크에서 연결 리스트

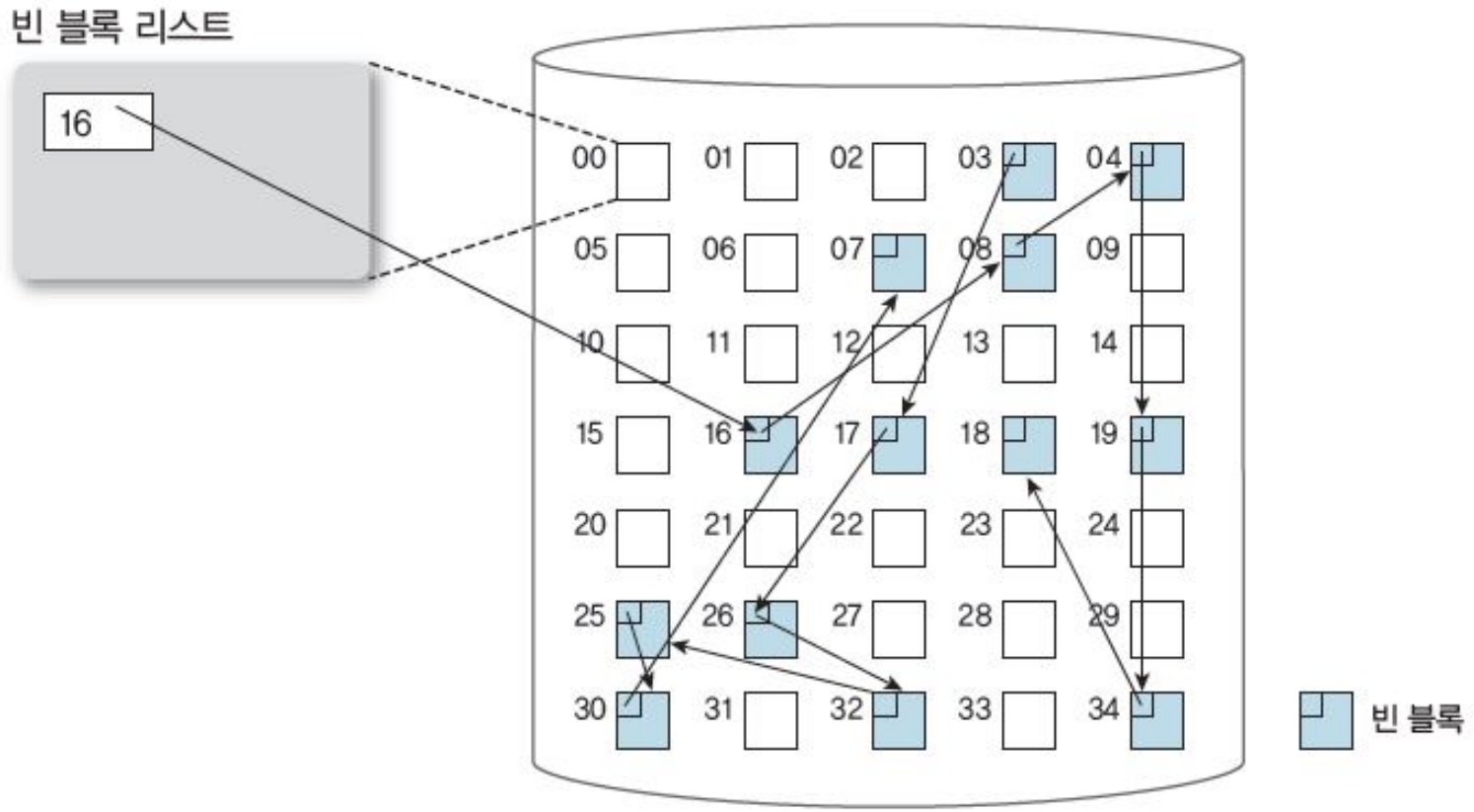


그림 10-29 **디스크에서 연결 리스트**

3. 디스크의 빈 공간 관리 방법

인덱스 블록(그룹핑)

- 빈 블록의 포인터를 인덱스 블록에 보관하며, 이들은 서로 연결
- 빈 블록 주소를 포함하여 인덱스 역할을 하므로, 첫 번째 빈 블록 내에 빈 블록 n개의 번지 저장
- 이것의 처음 n-1개는 실제로 비어 있고, 마지막 한 개는 다른 빈 블록 n개의 주소 포함하는 다른 블록의 디스크 주소
- 사용 가능한 블록 주소를 여러 개 쉽게 찾을 수 있게 함

3. 디스크의 빈 공간 관리 방법

- 인덱스 블록(그룹핑)

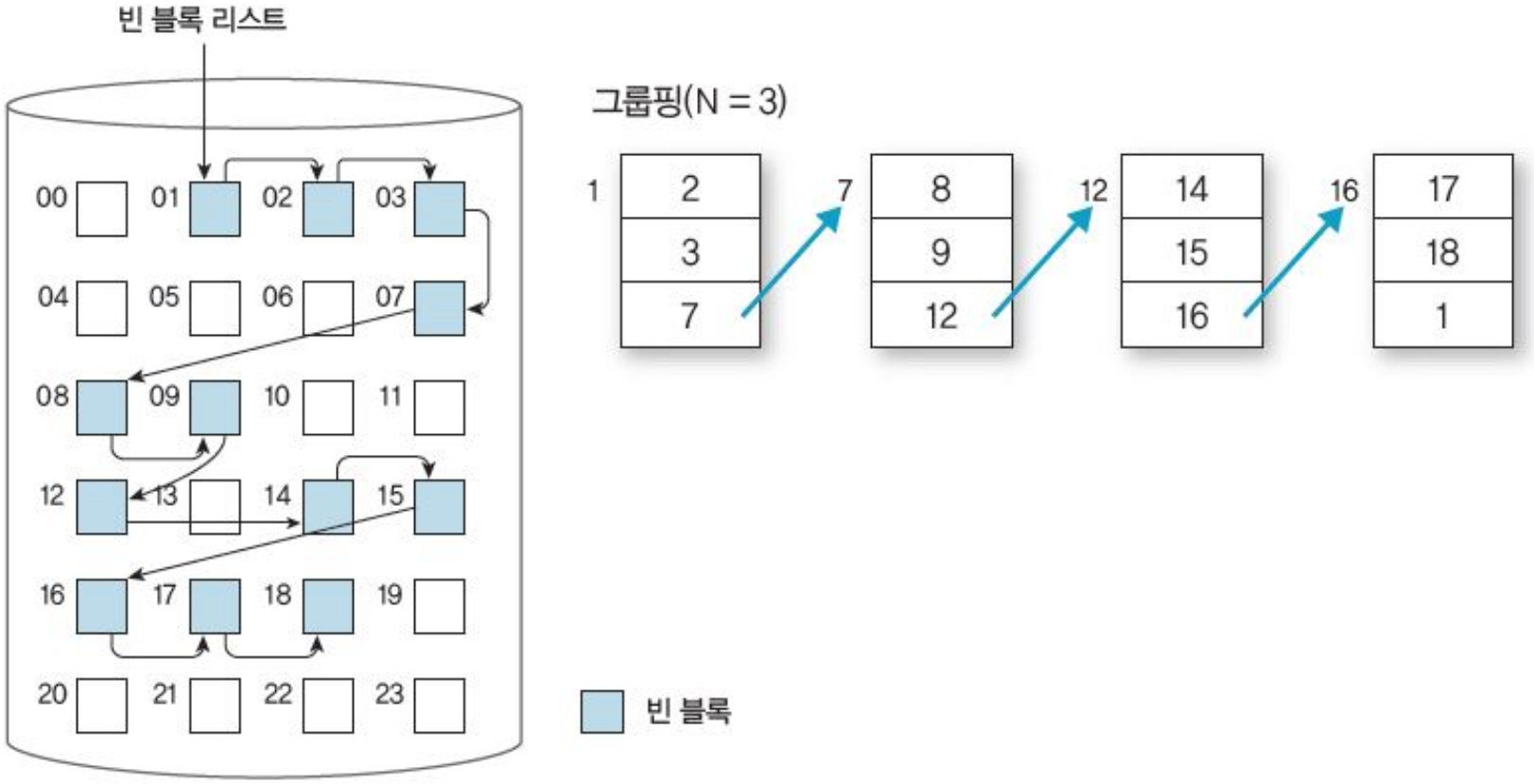


그림 10-30 인덱스 블록(그룹핑)

Section 05 파일 보호(1. 파일 보호의 필요성)

파일 보호의 필요성

- 컴퓨터 시스템은 다수의 사용자가 사용하므로 정보를 저장한 파일은 물리적인 손상이나 부적합한 액세스에서 보호해야 함. 특히 파일을 물리적인 손상에서 보호(신뢰성 보호)해야 함.
- 보통은 디스크에서 테이프로 파일을 자동 복사하여 파일이 불의의 사고로 손실되거나 손상되는 것을 방지
- 파일 시스템은 하드웨어의 문제점인 전원 장애, 헤드 파손, 먼지, 온도, 고의적인 파괴 행위로 손상을 받거나 우연한 사고로 제거될 수 있음
- 파일 시스템 소프트웨어의 오류로 파일 내용을 분실할 수도 있다. 단일 사용자 시스템이라면 단순히 사용 디스켓을 잘 관리하거나 잠금 장치를 이용하면 되지만 다수 사용자 시스템에서는 다른 방법이 필요

2. 파일 보호 방법

파일 명명file naming

- 액세스하려는 파일을 명명할 수 없는 사용자를 액세스 대상에서 제외하여 파일 보호 가능.
- 파일 이름은 보통 기억하기 쉬운 글자를 많이 사용하므로 추측도 가능

암호password(비밀번호)

- 각 파일에 암호를 설정하여 보호
- 암호를 임의로 정하고 자주 변경하면 파일의 액세스를 효율적으로 제한 가능
- 파일별로 독립된 암호가 있어 기억해야 할 암호가 너무 많아 비현실적
- 모든 파일의 암호가 하나이면 한 번만 노출되어도 모든 파일에 액세스할 수 있다는 문제
- 세밀한 수준의 보호를 제공하려면 다중 암호를 사용

액세스 제어access control

- 사용자에 따라 액세스할 수 있는 파일이나 디렉터리 리스트를 두어 사용자 신원에 따라 서로 다른 액세스 권한 부여
- 액세스 요구가 발생하면 운영체제가 이 액세스 리스트를 참조하여 액세스 여부 결정 방법으로 보호

2. 파일 보호 방법

액세스 그룹access group

- 액세스 리스트에서 주된 문제점은 리스트 길
- 모든 사용자에게 파일 액세스를 허용하면 모든 사용자가 읽기 액세스를 할 수 있도록 등록해야 하는 문제 발생
- 시스템에 있는 사용자의 리스트를 계속 추적하여 액세스 리스트를 구성하면, 고정 크기인 디렉터리 항목이 가변 크기로 바뀌는 등 복잡한 공간 관리 문제 발생.
- 문제 해결 방법
  - 소유자 : 파일을 생성한 사용자
  - 그룹 : 파일을 공유하고 비슷한 액세스가 필요한 사용자의 집합
  - 모든 사람 : 시스템에 있는 모든 다른 사용자
- 파일을 보호하는 그룹별 제어 비트

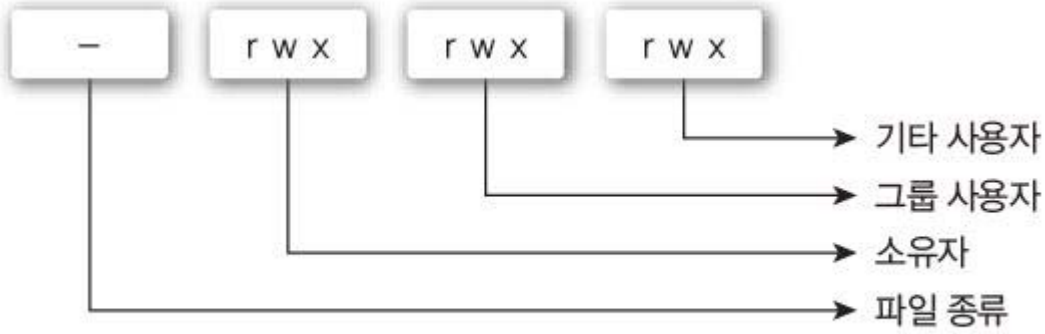


그림 10-31 파일을 보호하는 그룹별 제어 비트

2. 파일 보호 방법

■ 사용자 권한user permission 지정

- 제2의 방어벽, 침입자의 손상 정도를 줄일 수 있는 방법
- 사용자가 계정을 받을 때부터 특정한 디렉터리와 파일만 액세스할 수 있도록 시스템 관리자가 허락, 그 외의 영역은 액세스를 불허
- 네트워크와 호스트 컴퓨터 시스템 관리자의 사용자 권한 지정

표 10-2 **사용자 권한 지정**

권한	설명
삭제하기	파일 삭제를 허용한다.
생성하기	새로운 파일 생성을 허용한다.
쓰기	파일에 새 정보를 저장하는 것을 허용한다.
읽기	파일을 열어 정보를 읽는 것을 허용한다.
검색하기	권한 검색 명령을 이용하여 파일을 검색하는 것을 허용한다.