파일 시스템

Background Knowledge

- **File**은 의미있는 정보들을 담은 논리적인 저장 단위로 일반적으로 하드디스크, USB 같은 보조기억장치에 저장된다.
- **File attribute** 또는 **File metadata**는 파일을 관리하기 위한 각종 정보들이다. 파일 이름, 유형, 저장된 위치, 파일 사이즈 등의 정보가 해당된다.
- **File Descriptor**는 파일을 관리하기 위한 메타데이터를 보관하는 FCB(File Control Block)으로, 파일마다 독립적인 파일 디스크립터를 가진다.

File System

파일 시스템의 기능/목적

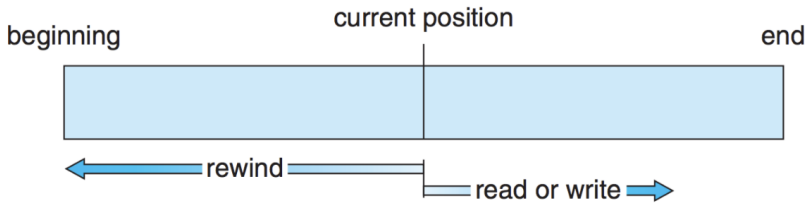
- 파일의 CRUD 기능을 수행한다.
- 파일 관리, 보조 저장소 관리, 파일 무결성 매커니즘, 접근 방법을 제공한다.
- 하드디스크와 메인 메모리의 속도차를 줄이고, 하드디스크 용량을 효율적으로 이용할 수 있도록 한다.
- 여러 사용자가 파일을 공유하여 사용하거나, 파일을 백업, 복구하는 기능을 제공한다.

Access Methods

파일 정보에 접근 방식에 따라 순차 접근, 직접 접근, 색인 접근이 있다.

순차 접근 Sequential Access

- 가장 단순한 방법. 파일의 정보가 레코드 순서대로 처리된다.
- 현재 위치에서 읽거나 쓰면 offset이 자동으로 증가하고, rewind하여 뒤로 돌아갈 수 있다.



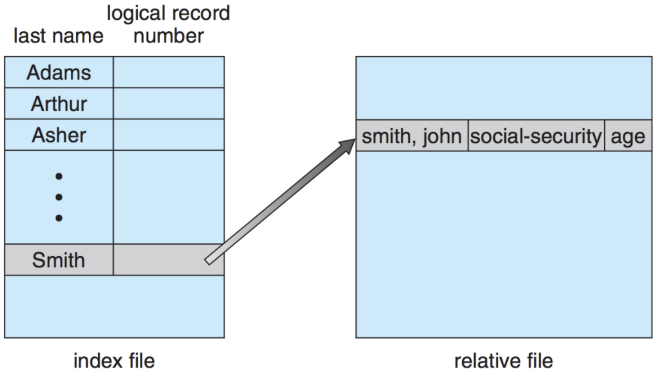
직접 접근 Random Access

- 읽기 쓰기 순서에 제약이 없고, 파일의 레코드를 임의의 순서로 접근할 수 있다.
- 대규모 정보에 즉각 접근하는 데에 유용하기에 데이터베이스에 이용된다.

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

색인 접근 Index Access

- 파일에서 레코드를 찾을 때 색인을 먼저 찾고 대응되는 포인터를 얻어서 접근한다.
- 크기가 큰 파일에서 유용하다.



Directory

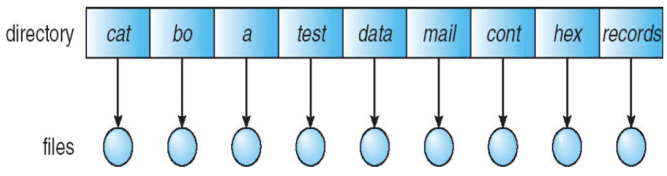
디렉터리는 파일의 메타데이터 중 일부를 보관하고 있는 일종의 특별한 파일이다. 디렉터리는 다음과 같은 기능을 제공한다.

- 파일 찾기(Search), 생성(Create), 삭제(Delete)
- 디렉터리 나열(List)
- 파일 재명명(Rename)
- 파일 시스템 순회(Traverse)

디렉터리의 논리적 구조를 정의하는 여러 방법이 있다.

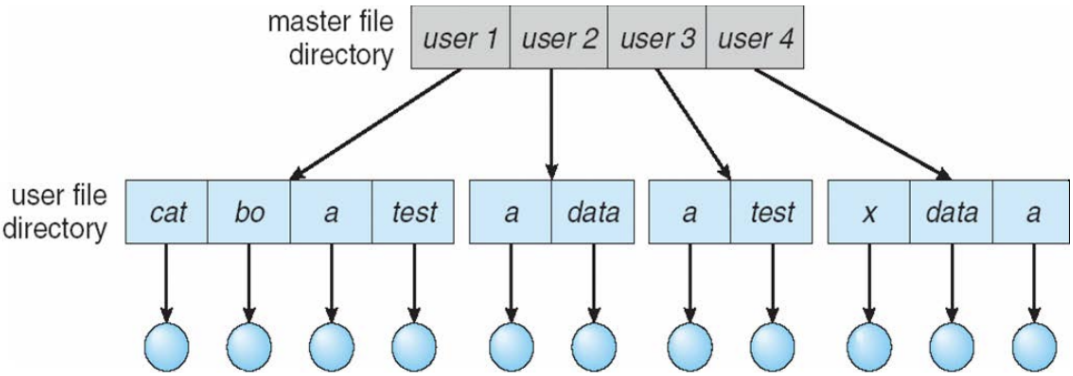
1단계 디렉터리

모든 파일이 디렉터리 밑에 존재하는 형태로, 파일은 유일한 이름을 가진다. 쉽고 이해하기 편한 방식이지만 파일이 많아지거나 다수의 사용자가 사용하는 시스템에서 같은 이름을 허용하지 않는 것은 큰 제약이다.



2단계 디렉터리

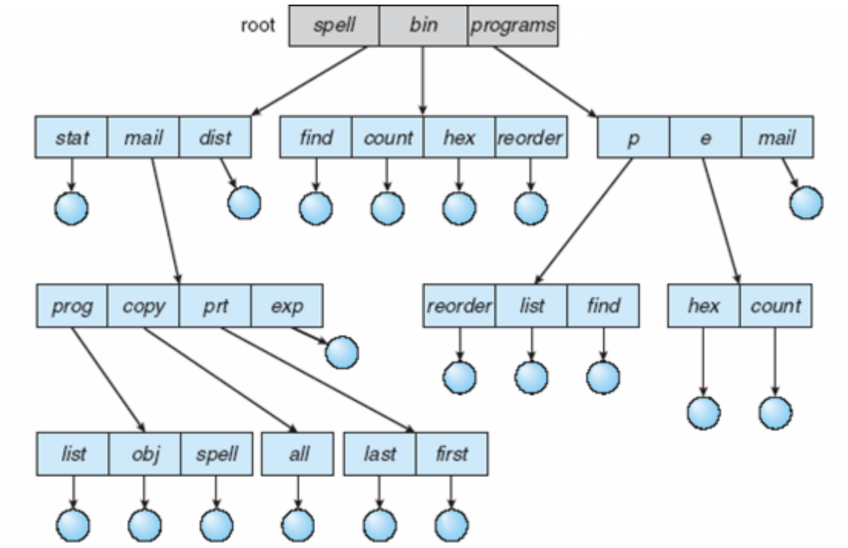
중앙에 마스터 파일 디렉터리가 있고, 그 아래에 각 사용자별 별도의 디렉터리를 두는 형태다. 서로 다른 사용자라면 같은 이름의 파일을 가질 수 있다. 하지만 각 사용자 디렉터리가 독립적이므로 파일을 공유할 수 없고, 다른 사용자의 파일에 접근해야 하는 경우 단점이 된다.



트리 구조 디렉터리

하나의 루트 디렉터리 아래에 여러 개의 서브 디렉터리가 있는 구조. 모든 파일은 고유한 경로를 갖게되며 효율적인 탐색과 그룹화가 가능하다.

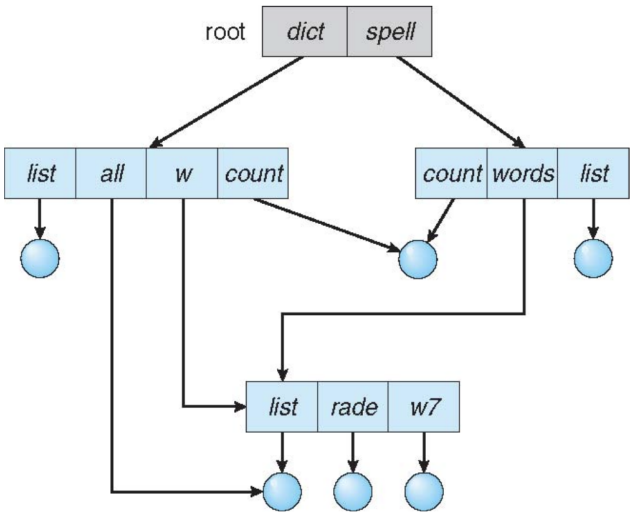
디렉터리는 일종의 파일이므로 일반 파일인지 디렉터리인지 구분할 필요가 있다. 이를 구분할 수 있는 비트를 두고 0이면 일반 파일, 1이면 디렉터리로 구분한다.



비순환 그래프 디렉터리

하위 파일이나 디렉터리의 공유를 허용한다. 단 사이클이 존재하지 않는다. 하나의 파일이나 디렉터리가 여러 경로를 가질 수 있고, 공유된 파일을 탐색하는 경우 다른 경로로 여러 번 찾아갈 수 있어 이러한 경우 시스템 성능이 저하된다.

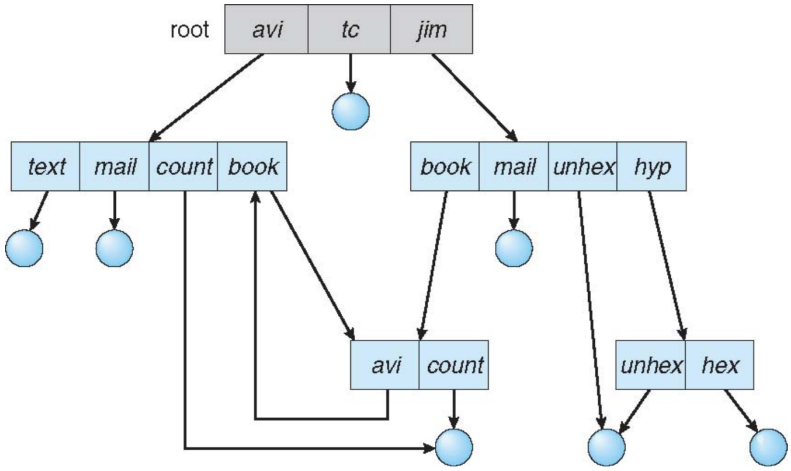
공유된 파일이 삭제되는 경우 파일은 삭제되었지만 파일을 가리키는 포인터가 남아있는 dangling pointer가 발생할 수 있다.



그래프 구조 디렉터리

파일이나 디렉터리를 공유할 수 있고, 사이클도 허용한다.

불필요한 파일 제거를 위한 참조 counter가 필요하고, 제거된 파일의 디스크 공간 확보를 위해 쓰레기 수집이 필요하다.



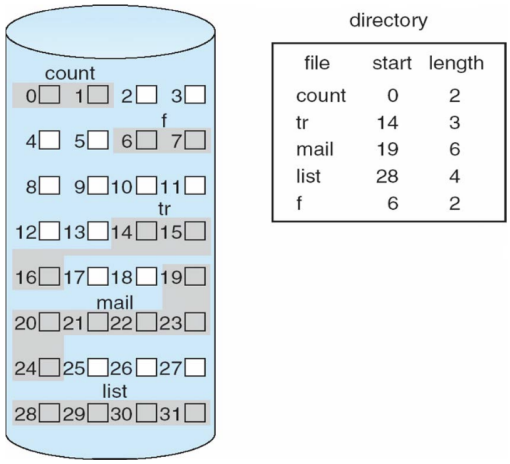
Allocation Method 파일 할당 방식

디스크 내에 여러 파일이 저장되는데 파일을 어떻게 배치할 것인가

Contiguous Allocation

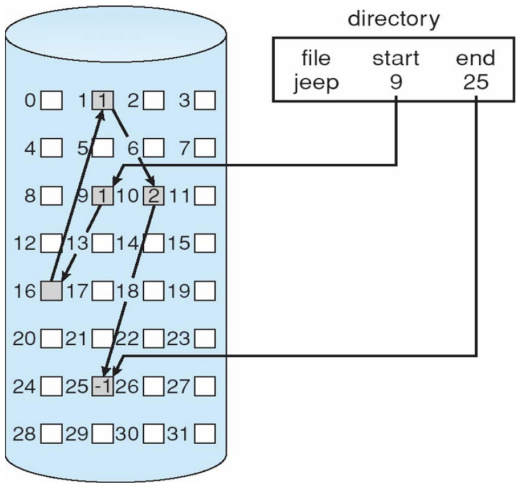
- 파일을 디스크에 연속적으로 할당
- 디렉터리에 파일의 시작 위치와 파일의 길이를 저장하여 탐색할 수 있다.
- 연속적으로 저장되어 random access이 가능하다.

- 파일을 지우고 쓰고를 반복하면서 external fragmentation 발생. fragmentation을 없애는 compaction 과정이 필요하다.
- 파일의 크기를 키우기 어렵다. 파일이 커지는 것을 고려해서 미리 큰 공간을 할당하면 internal fragmentation이 발생할 수 있다.



Linked Allocation

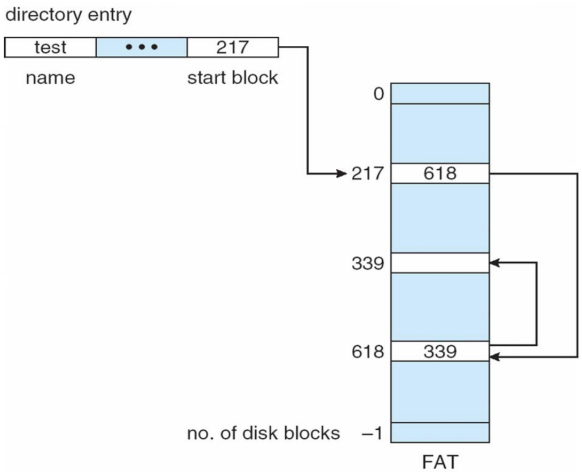
- 파일을 블록 단위로 쪼개서 linked list 형태로 관리한다. 빈 위치에 자유롭게 할당하고 포인터로 다음 위치를 가리킨다.
- 디렉터리에는 시작하는 위치와 끝나는 위치를 저장한다.
- sequential하게 순서대로 접근하는 것은 상관없지만 i-th 블록을 찾으려면 첫 인덱스부터 탐색해야하는 단점이 있다. Random access가 불가능하다.
- 포인터가 블록 공간 일부를 차지한다.
- 포인터 하나만 유실되어도 많은 정보를 잃게 되는 신뢰성 문제도 있다.



FAT (File Allocation Table)

- FAT은 파일이나 디렉토리의 위치 정보(포인터)를 저장하는 테이블
- FAT에서 블록 위치를 찾을 수 있으므로 Random access 가능하다.

- 포인터를 별도의 공간에 저장하여 신뢰성 문제를 해결한다.



Indexed Allocation

- 한 블록에 파일에 대한 인덱스를 모두 저장한다.
- External fragmentation이 발생하지 않고 random access가 가능하다.
- 작은 파일을 저장할 때도 인덱스 블록 하나를 사용해야하므로 공간 낭비가 있다. 아주 큰 파일은 블록 하나에 인덱스를 모두 저장할 수 없는 문제가 발생할 수 있다.