1. 일반 파일 시스템

1. File Concept

- 운영체제는 컴퓨터 시스템을 편리하게 사용하기 위해 저장된 정보에 대한 일과된 논리적 관점을 제공하고 저장 장치의 물리적 특성을 추상화하여 논리적 저장 단위, **파일**을 정의한다.
- 파일의 속성에는 이름, 식별자, 타입, 위치, 크기, 시간 등 이 있다.
- 파일 명령어에는 생성, 쓰기, 읽기, 재설정, 삭제, 절단(내용만 지우고 속성은 남김)이 있다.
- 파일 연산을 하기 위해 open() 시스템 호출로 파일을 접근한다. 최초 파일 접근시 재접근에 대한 낭비를 줄이기 위해 open-file table에 모든 열린 파일에 대한 정보가 저장된다.
- 파일이 더이상 사용되지 않으면 프로세스에 의해 닫히고 open-file table에서 제거된다. - close()
- 파일 포일터 : 읽기와 쓰기 시스템 호출의 일부분으로 파일을 어디까지 읽었다는 것을 나타낸다.
- File-open count : 파일에 대해 몇개의 프로세스가 접근했는지 나타낸다.
- Open File Locking : Shared lock 여러 프로세스가 잠금을 할 수 있다. writer lock 한번에 한 프로세스만 잠금을 할수있다.

메모리 관리 기법과 파일 시스템은 모두 기억 장치를 관리한다. 기억 장치는 RAM이건 하드 디스크이건 한정되어 있는 자원이기 때문에 최대한 공간을 아껴서 사용해야 한다. 그렇기 때문에 메모리 관리 기법과 파일 시스템 모두 내/외부 단편화를 최소화 하기 위해 노력해야 한다. 그렇다면 메모리 기법과 파일 시스템 간의 차이점은 무엇일까?

해답은 **이름** 이라는 특성이다. **‘이름’**이라는 특성을 제외하곤 메모리 관리 기법과 파일 시스템은 같은 소프트웨어이다. 바로 ‘이름’을 입력으로 받아 해당 데이터를 리턴해주는 소프트웨어가 파일 시스템이다. 파일 시스템이 하드 디스크에 저장하는 정보는 크게 **Meta data**와 **User data**로 나뉜다.

(1) Meta data: 파일의 속성 정보나 데이터 블록 인덱스 정보 등이 해당된다.
(2) Users data: 사용자가 실제 기록하려 했던 내용이 저장된다.

2. 디스크 구조와 블록 관리 기법

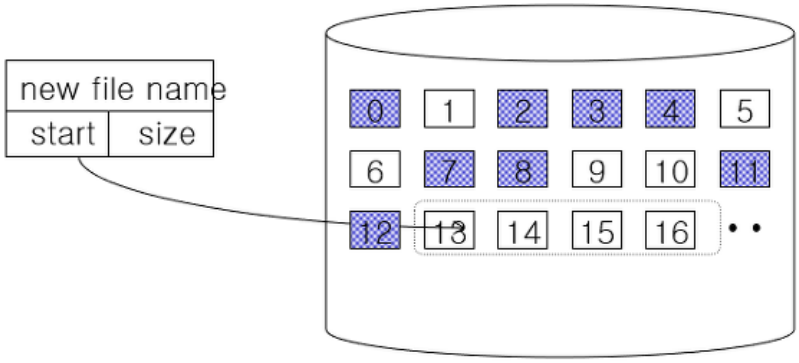
디스크에서 데이터를 접근하는 데 걸리는 시간은 탐색시간, 회전 지연 시간, 데이터 전송 시간 이라는 세가지로 구성된다.
(1) 탐색시간(seek time): 헤드를 요청한 데이터가 존재하는 트랙 위치까지 이동 시간
(2) 회전 시간(rotational latency): 요청한 섹터가 헤드 아래로 위치될 때 까지 디스크 원판을 회전시키는 데 걸리는 시간
(3) 데이터 전송 시간(transmission time): 헤드가 섹터의 내용을 읽거나 또는 기록하는 데 걸리는 시간

하지만 파일 시스템은 디스크를 물리적인 구조로 보지 않고 논리적인 디스크 블록들의 집합으로 본다.

디스크 블록은 0, 1, 2 등의 논리적인 번호를 하나 씩 갖는다. 그리고 디스크 블록의 크기는 일반적으로 페이지 프레임의 크기와 같다.(4KB) 사실 파일 시스템 성능의 최대 병목 요소는 디스크 I/O 이다. 디스크 블록의 크기가 클수록 한 번의 입출력으로 더 많은 데이터를 메모리로 읽어 들일 수 있기 때문이다. (공간효율성 안좋음) 디스크 블록을 할당하는 방법에는 크게 연속 할당과 불연속 할당 두 가지 방법이 있다.

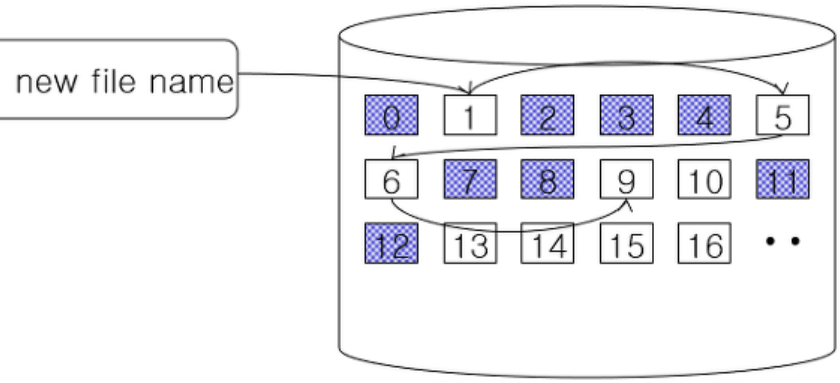
14KB의 파일 생성을 요청받았다고 가정했을 때 어떤 방법으로 디스크블록을 할당할까?
불연속 할당 기법에는 블록체인, 인덱스 블록, FAT 기법 등이 있다.

(1) 연속할당



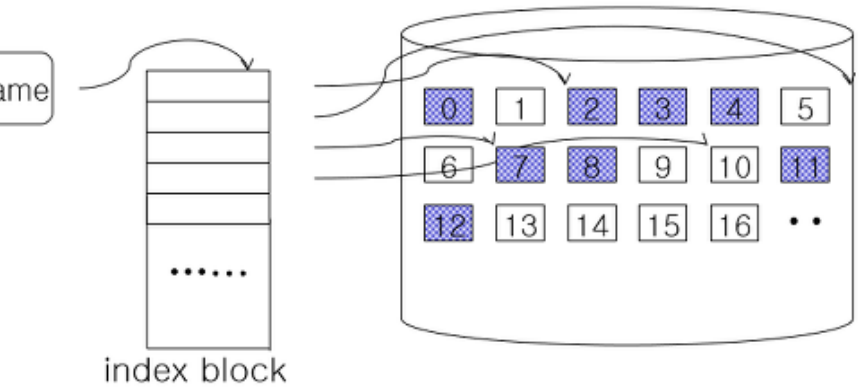
연속할당은 불연속 할당에 비해 파일을 읽는 속도가 빠르다. 왜냐하면 연속적으로 디스크 블록이 이루어져서 디스크의 탐색 시간을 줄일 수 있기 때문이다. 하지만 연속할당의 경우 파일의 크기가 변하면 문제가 될 수 있다.
가령 14KB -> 17KB로 파일의 크기가 커질 경우 4개가 아니라 연속의 5개의 디스크 블록이 필요하기 때문이다. 결국 연속 할당을 위해서는 기존에 있던 디스크 블록들을 더 넓은 곳으로 복사한 후 새로운 내용을 추가해야 한다. 이는 성능 상 매우 큰 문제가 된다.
따라서 파일시스템을 설계할 때 연속 할당 방법만을 사용하는 경우는 거의 없다.

(2) 블록체인 할당



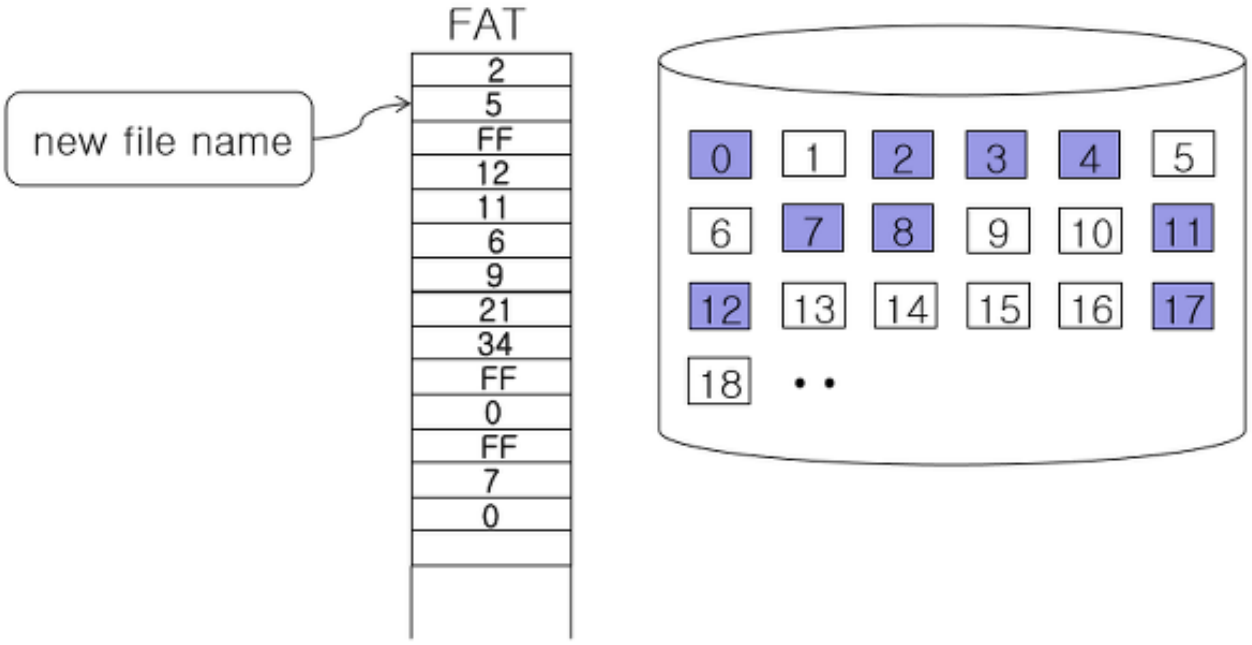
블록체인 기법은 같은 파일에 속한 디스크 블록들을 체인으로 연결해 놓는 방법이다. 따라서 특정 파일에 속한 첫 번째 디스크 블록에 가면 포인터를 이용해 다음 블록의 위치를 찾아갈 수 있다. 그러나 이 경우 lseek()같은 시스템 콜을 사용하여 파일의 끝 부분을 읽으려는 경우에는 어쩔 수 없이 앞부분을 읽어야 한다. 또한 중간 부분이 유실될 경우 나머지 데이터 모두 잃게 된다는 단점이 있다.

(3) 인덱스 블록 할당



인덱스 블록 기법은 블록들에 대한 위치 정보들을 기록한 인덱스 블록을 따로 사용하는 방법이다.
이 방법에서는 파일의 끝 부분을 접근하는 경우 데이터 블록을 일일이 읽어야 한다는 단점은 없지만, 만약 인덱스 블록이 유실되면 파일의 데이터 전체가 소실되는 문제가 있다.
또한 인덱스 블록을 위한 별도의 공간이 필요하며 파일이 커져 인덱스 블록이 가득 찰 경우 이를 해결하는 방법이 필요하다.

(4) FAT 할당 기법



FAT 기법은 같은 파일에 속해 있는 블록들의 위치를 FAT 이라는 자료구조에 기록해 놓는 방법이다.
인덱스 블록 기법은 파일마다 인덱스 블록이 필요한데, FAT 기법에서는 파일시스템 전체적으로 하나의 FAT 이 존재하는 것이다.
FAT 구조에서 FF는 파일의 끝을 의미하여 0은 free상태를 나타낸다.
한편 FAT 구조의 유실은 파일 시스템 내의 모든 파일이 소실된다는 문제가 있다. 따라서 요즘 대부분의 FAT 구조 파일 시스템은 FAT 내용을 중복하여 관리한다.