운영체제_Hard Disk Drives

기본 단위는 Sectors 로서 512 Byte정도의 크기를 가진다. 하드디스크는 이 섹터들의 집합(Array) 라고 생각하면 된다. 단, 이 어레이가 선상의 것이 아닌 원형의 것이라고 생각하면 된다. 한 섹터를 작성할 때에는 Atomic하고 이것을 보장한다. 10 byte쓰고 502남고 그런 것이 없다는 말이다.

- Torn write

  If an untimely power loss occurs, only a portion of a larger write may complete.

섹터들을 동심원 으로 배치한 것이 트랙이라고 한다. 이 트랙은 매우 빠른속도로 돌며 데이터를 읽거나 쓰게 된다. 전기장 신호를 자기장 신호로 변경하며 이용하게 되는데 이 일을 하는것이 Disk head라고 한다.

이 head를 섹터위에 올려놓고 움직이며 일을 한다.

위와같이 여러개의 판을 가지고 이용하기도 한다.

Rotational delay : 읽으려는 섹터를 위해 트랙을 얼마만큼 돌아서 이동해야 하는지에 대한 딜레이.

Seek time : 트랙에서 트랙으로 이동하게 되는 시간을 뜻한다.

Transfer

데이터를 읽거나 쓰는데 걸리는 시간.

I/O 가 완료될때 까지의 타임은 seek + rotational delay + transfer이다.

그런데 seek하는 도중에 트랙도 돌고 있는 상태이기 때문에 이것을 고려해서 시간을 측정해야 한다. (HW-Disk참조)

Track Skew

현재 트랙에서 다 읽은 뒤 다음 트랙에서 읽으려고 할때 트랙을 넘어가는 시간 때문에 섹터가 넘어가버려서 읽지 못하는 상황이 발생 하는데 이 때문에 섹터를 두번 정도 뒤로 밀어서 여유를 두어 읽을 수 있도록 한다.

Cache (Track Buffer)

캐쉬 사용에서의 항상 문제는 write에 대한 것인데, write를 해서 바뀐 데이터를 언제 반영할 것이냐에 대한 문제가 생긴다.

- Write back

  바뀐 것이 생기면 즉시 변경한다. 성능이 안좋아지는 문제.

- Write through

  바뀐 것을 모아서 변경한다. 중간에 오류나 깨지는 등의 문제가 생기면 해결하기 어렵다.

Disk Scheduling

IO처리에 대한 명령을 쌓고 그것을 컨트롤러에게 보내 처리하게 된다. 그런데 이때 어떠한 명령을 먼저 처리할 것이냐에 대한 문제가 나타난다. (FIFO, SSTF 등) 자세한 스케줄에 대한 처리는 컨트롤이 결정하는 추세이다.

- SSTF(Shortest Seek Time first)

  Seek 타임이 최대한 줄어드는 방향으로 스케줄링 해서 처리하는 방법. 하지만 이 방법은 꾸준이 요청이 들어온다면 끝까지 도달하지 않을수도 있다는 문제가 존재한다(**Starvation**) 또한 단순히 인덱스만으로는 트랙이 바뀌는지 바뀌지 않는지 바로 알기 힘드므로 NBF 를 사용한다.(Nearest block first - 가까운 것에 대한 우선 처리) 이를 사용할때에도 문제가 발생할 수 있다.

- SCAN - Elevator 알고리즘

  바깥쪽에서 안쪽으로, 안쪽에서 바깥쪽으로 쭉 가면서 처리하는 방식. 요청이 존재하던 하지 않던 끝까지 가는 방식. 이는 Starvation 현상을 해결할 수 있다. 그런데 상대적으로 볼때 가운데 있는 것이 바깥보다 더 자주 처리하게 되는 형평성의 문제가 존재한다.

  - F-SCAN(Freeze SCAN)

    Freeze the queue to be serviced when it is doing a sweep Avoid starvation of far-away requests

  - C-SCAN(Circular SCAN)

    Sweep from outer-to-inner, and then reset at the outer track to begin again A bit more fair to inner and outer tracks

SPTF(Shortest Positionning Time first) or SATF(Shortest Access Time first)

SSTF가 rotation delay를 고려하지 않는점을 보완한 것. Rotation Delay까지 고려해서 스케줄링을 수행한다.

File System

파일시스템은 크게 두가지 File 과 Directory로 구분한다. 파일은 선형의 어레이에 정해진 바이트상에 위치한다.

NTFS ,FAT 16등 이것은 파일시스템의 이름이다. 우리가 이것들을 이용하고 있다. 이것을 어떻게 구성하느냐 그런 문제가 있다. 우리는 VSFS(very simple filee system)만을 배운다.

어떠한 설명하는 정보를 우리는 메타 정보라고 한다.

각 파일에는 이것의 이름이 존재한다.(path를 포함한) 메모리의 페이지 크기가 통상 4k 이기 때문에 블럭의 크기를 4k에 맞춘다. 섹터의 크기만큼 블럭의 크기를 잡지 않는 이유는 이렇게 잡으면 너무 빈번하게 읽고 쓰고 하기 때문에 비효율 적이다.

파일을 구성할 때 이 파일이 어느블럭 어느블럭 으로 구성 되어있고 이것은 어디 어디에 있다 라는 사항을 알아야 한다.

파일을 모았다가 한번에 요청을 보낸다면 스케줄링을 재수행 하기 때문에 좀더 빠르게 수행할 때도 있으며, 컴파일상 쓸 필요가 없어지는 경우도 발생하기 때문에 딜레이를 조금 둔다. (5~30초) 하지만 딜레이 되면 안되는 것도 존재 하는데 (은행과 같은 것 등) 딜레이 되는동안 갑자기 꺼지거나 한다면 금전적인 중요한 문제가 발생될 수 있기 때문에, fsync() 라는 것을 두어서 이용한다 .

Metadata

파일 관리에 필요한 파일 시스템 정보가 여기에 다 들어있다. 여기의 정보를 읽을 필요도 있고 바꿀 필요도 존재한다.

stat으로 메타데이터를 보거나 할 수 있다.

Link : 파일이라는 경로가 있는데 파일2라는 경로를 추가한다.

Symbolic link : 또다른 파일을 가르키는 포인터 (바로가기와 비슷)

mkfs같은 명령어로 파일 시스템을 시작할 수 있다.

헤더를 i-node라고 하고 인덱스를 i-number라고 한다. 그러면 헤더만 잘 찾으면 된다. 이 i-node를 어레이처럼 블락에 구성시켜서 이용한다.(헤더의 크기가 상대적으로 작기 때문에 이렇게 이용한다.) 헤더를 뽑아 헤더에 적인 블락위치들로 가서 데이터를 꺼내와서 이용한다.

Very simple file system

1섹터는 512-byte 섹터단위로 쓸때 아토믹 한데, 8개의 섹터를 1블럭으로 잡는다. i-node에 대한 것을 앞쪽에 두고 뒤쪽에 데이터를 놓는 형식으로 이용한다.

우리가 어떤 것을 저장할 때 i-node와 데이터를 저장 할 것인데, 빈 곳이 어딘지 먼저 알아야 저장을 할 수 있다. 어차피 사이즈가 같은 것들이 있으므로 사용중인지 아닌지를 비트로 표기한다. 맨앞 두블록정도 그렇게 이용한다. 작은것이 대부분이므로 일단 작은 파일들을 기준으포 크기를 맞추고 더 큰파일에 대해서는 몇번의 경로를 추가하는 indirect방식을 사용해서 용량을 늘려서 나타낸다.

그런데 아이노드 관련 정보를 읽을때 필요 아이노드 하나만 읽을 방법이 없다. 다른 아이노드들 까지 한번에 다 읽게 된다. 쓸때도 통채로 다 읽은뒤 쓰는 행위가 필요하다.

파일 한번 쓰기위해 굉장히 많은 IO요청을 처리하도록 되어있다.

프로그램 - HW

Inode bitmap의 비트를 가지고 현재 inode가 사용중인지 아닌지를 알수 있다. a가 -1이라는 것은 어드레스가 없다는 즉 아직 데이터를 가지지 않고 있다는 뜻이다. r은 레퍼런스 카운터라고 해서 갈수있는 레퍼런스가 몇개인지를 나타난다.