

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – GJ (박현우)
uc820@naver.com

목차

Chapter 5 파일시스템과 가상 파일시스템

- 1) 파일시스템 일반
- 2) 디스크 구조와 블록관리 기법
- 3) FAT 파일시스템
- 4) Inode 구조
- 5) Ext2 파일시스템
- 7) 가상 파일시스템 (Virtual File System)
- 8) 태스크 구조 vs VFS 객체

Chapter 5 파일 시스템과 가상 파일 시스템

* 파일은 디스크의 추상화 시켜서 프로그램에게 잘수를 부여한다.

1) 파일 시스템 일반

1) 파일 시스템은 무엇인가?

⇒ 보조 기억 장치 (하드 디스크) 라고 불리는 저장 장치를 관리해 주는 소프트웨어이다.

2) 파일 시스템을 사용하는 이유는 뭘까?

⇒ 기억 장치는 RAM이건 하드 디스크이건

한정된 자원을 아껴 써야 한다. 그래서, 메모리 관리된 파일 시스템 모두 내/외부 단편화의 최소화를 위해 노력해야 한다.

3) 메모리 관리 기법과 파일 시스템 간의 차이는 무엇인가?

⇒ '이름'이라는 특성을 빼면 유사하다. '이름'을 입력으로 해당 데이터를 리턴해 주는 소프트웨어가 '파일 시스템'이다.

4) 이러한 기능을 제공하기 위해 파일 시스템은 하드 디스크에 어떤 내용을 저장할까?

⇒ 하드 디스크에 meta data와 사용자 데이터를 저장한다.
inode, super block 파일 내용

ex) 사용자가 파일 하나 생성 → 파일 시스템은 디스크 블록을 하나 할당 받음 → 기록

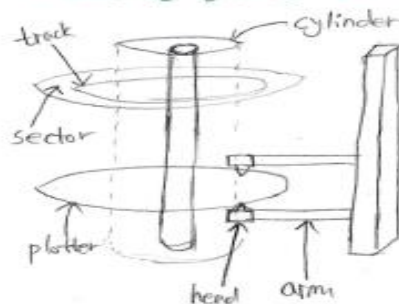
↓
device driver가 할당해준다.

5) 나중에 파일을 읽으려 면?

⇒ 파일 시스템은 데이터를 추상 자원인 '파일'로 제공하기 때문에 추가 정보를 기록해 둔다.
(파일 이름, 인덱싱 정보)

2) 디스크 구조와 블록 관리 기법

* 파일 시스템이 저장 공간의 기본 접근 단위인 '디스크 블록'을 관리하는 기법에 대해 알아보자.



Sector: 디스크에서 데이터를 읽거나 기록할 때 기본 단위 (512 byte)

Cylinder: 원판에서 같은 위치를 갖는 track들이 집합

1) 디스크의 전체 용량은 어떻게 결정되나?

⇒ 트랙마다의 섹터 개수로 결정된다.

2) 하드 디스크의 용량과 같은 정보는 누가 저장하고 있나?

⇒ device driver가 가지고 있다.

→ 하드 디스크는 자기장으로 정보를 입력하기 때문에 device driver에는 자기장을 읽는 연산 알고리즘도 필요하다.

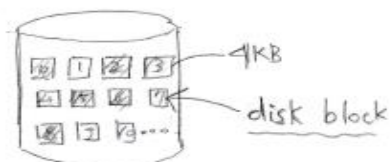
3) 디스크에서 데이터를 접근하는데 걸리는 시간은?

① 탐색 시간 : 요청한 데이터가 존재하는 트랙까지 (seek time) 헤드를 움직이는 시간

② 데이터 전송 시간 : 섹터의 내용을 읽거나 기록하는 (transmission time) 시간

③ 회전 시간 : 디스크 원판의 회전 시간 (rotational latency)

탐색 < 회전 < 전송 (빠르다)



디스크의 논리구조

4) 일반적인 파일 시스템은 왜 4KB disk block 을 가지나?

⇒ 페이지 프레임이 4KB이기 때문이다.

*문제는 없나?

⇒ 성능적인 병목요소인 디스크 I/O 문제가 있어 최근에는 디스크 블록 크기를 더 크게 하는 경향이 있다.

하지만, * 디스크 블록 크기가 ↑ 될수록 '쓰는 ↑'이지만 '공간 효율성'은 매우 떨어진다.

5) * 섹터와 디스크 블록은 무슨 관계인가? ∴

⇒ 디스크 블록의 크기는 4KB이고, 섹터의 크기가 512byte 라면 디스크 블록당 8개의 섹터가 대응된다.

6) 디스크 블록은 어떻게 읽히나?

⇒ 파일 시스템에서 요청을 받으면 8개의 섹터 내용이 읽혀진다. 디스크 블록 번호가 디스크 디바이스 드라이버에 의해 매핑이 되어져 있다.

7) 디스크 블록을 할당하는 방법은 무엇인가?

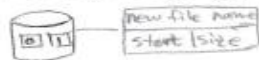
⇒ ① 연속 할당 : 탐색 시간이 짧다.

② 불연속 할당 : 공간 활용은 좋으나 탐색 시간이 길다

8) 디스크 조각 모음하면 불연속적인 부분을 연속의 부분으로 재정렬 한다.

8) 할당이 두 방법중 어떤 것이 파일 시스템

장점할까?



⇒ 연속 할당이 당연 파일 읽는 속도가 빠르다. 하지만, 14KB의 파일을 사용자가 내용을 추가 하여 17KB로 커졌다면 디스크 블록을 하나 더 늘려야한다. 그런데 디스크 블록이 쓰여지고 있다면 다른 공간을 찾아야 한다.

9) 다른 공간을 찾아 5개의 빈 공간으로 이동 후 복사해서 연속적으로 만들면 안되나?

⇒ 파일 크기 커질 때, 기존 블록을 옮기는 것은 성능상 매우 큰 문제가 있다. (복사)

결국은, 연속 할당만을 사용하는 파일 시스템은 없다.

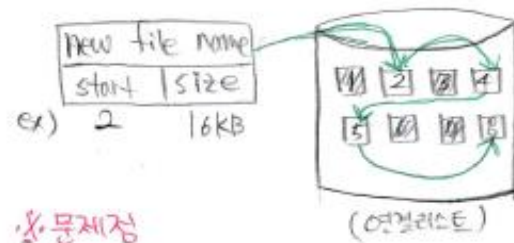
10) * 불연속 할당은 연속 할당과 달리 디스크 블록 어디에 위치하고 있는지에 대한 정보를 기록해야 하는데 무슨 방법이 있나?

⇒ ① 블록체인 기법

② 인덱스 블록 기법

③ FAT (File Allocation Table) 기법

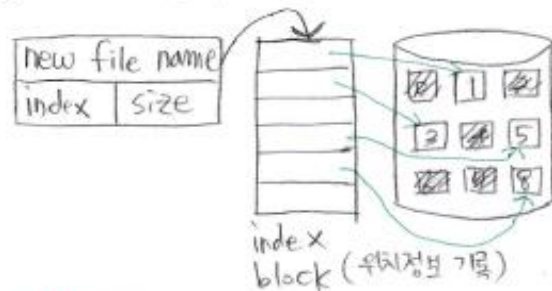
① 블록체인 기법



※ 문제점

첫번째 디스크블록에 가면 포인터를 이용해 다음블록에 갈 수 있다. 하지만 $lseek()$ 같은 시스템 콜을 사용하여 파일의 끝 부분을 읽으면 어쩔 수 없이 앞부분의 데이터 블록을 읽어야 함. 게다가, 중간 블록을 유실한다면 나머지 데이터를 모두 읽는다.

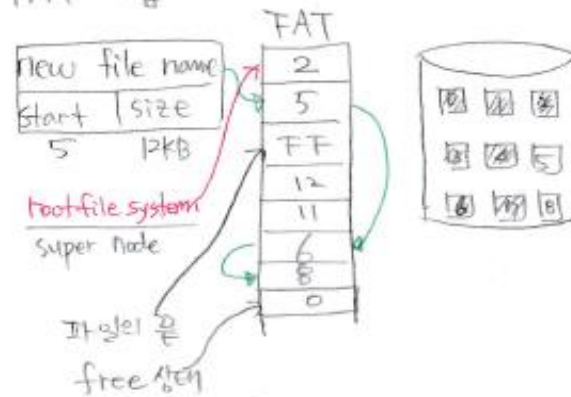
② 인덱스 블록 기법



※ 문제점

$lseek()$ 를 사용하여 끝부분을 접근 시 데이터를 블록을 일일이 읽어야 할 단점은 없지만, 인덱스 블록이 유실되면 파일 데이터 전체가 소실된다.

③ FAT 기법



※ 문제점

이 기법 또한 파일 중간을 읽기 위해서 처음부터 읽을 필요는 없으나, FAT 구조의 파일 파일시스템 내의 모든 파일이 소실되는 문제가 있다.

11) 리눅스에서의 파일시스템인 Ext2나 Ext4는 무슨 기법을 사용하나?

⇒ 인덱스 기법과 유사한 inode를 사용한다.

• Ext2는 용량 4Tera 한계.
데이터 복원 X

• Ext4 용량 무한대 ⇒ for Google Server
데이터 복원 O

3) FAT 파일 시스템

※ 리눅스에서 디렉토리나 파일은 별도 구분없이 파일로 취급된다. 결국 디렉토리는 자신의 이름을 가진 파일의 이름들을 데이터로 가진 특수 파일.

디렉토리 엔트리의 검색

1) 다른 파일시스템의 디렉토리 엔트리는 어떤가?

⇒ ① 저마다 자신의 디렉토리 구조체 선언

② 파일 생성 시, 이 구조체의 내용을 채워서 디스크에 저장

③ 나중에 사용자가 특정 이름의 파일 내용을 읽도록 요청하면

④ 파일이름을 가진 디렉토리 엔트리를 찾음

⑤ 디렉토리 엔트리가 가진 정보를 사용자에게 제공

ex) `int main(void){`

① `struct dirent *p;`

`DIR *dp;`

`dp = opendir(".");`

② `p = readdir(dp);`

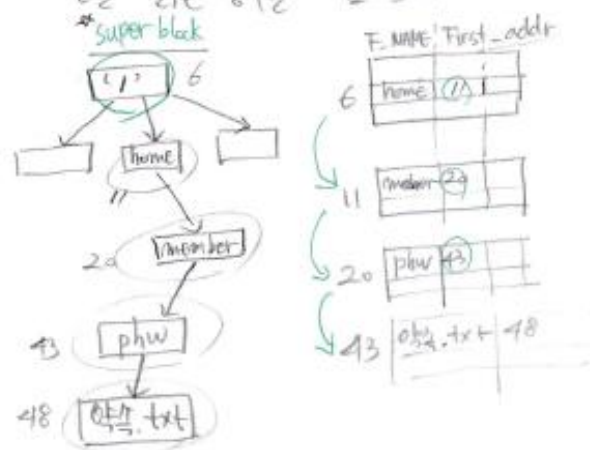
③ `p->d_name[0] == '.'`

2) 디렉토리 엔트리를 어떻게 찾나? 순차 탐색일까?

⇒ 순서가 가늠은 하지만 너무 느리다. 그래서 파일의 이름을 통해 접근한다.

먼저, 파일의 이름을 현재 디렉토리 위치를 기준으로 하는 상대경로와 '/' (root)에서 시작되는 절대경로 두 가지를 나눈다.

ex) 사용자가 `/home/member/phw/약속.txt` 라는 파일을 읽는 경우를 예로 보자.

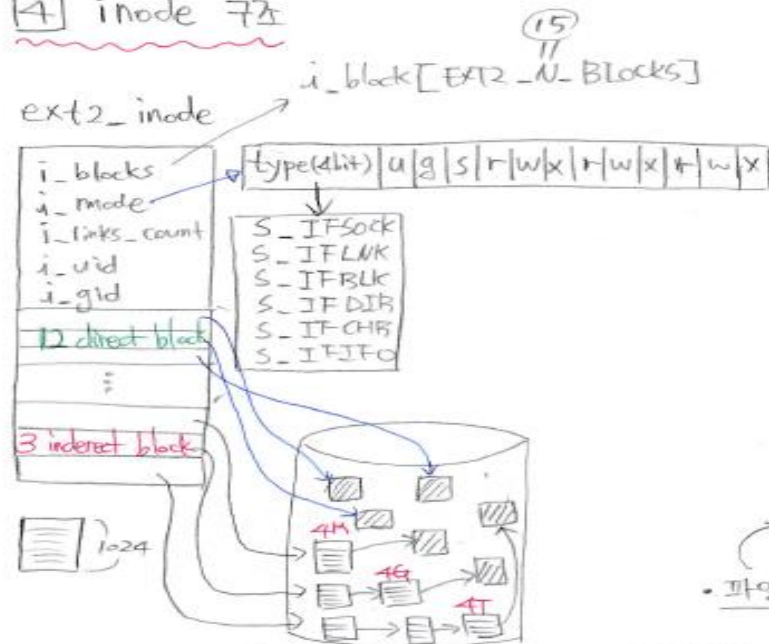


`struct ext2_sb_info {`

⇒ 파일 시스템의 전체적인 정보와 '/'의 위치를 기억.

}

4. inode 구조



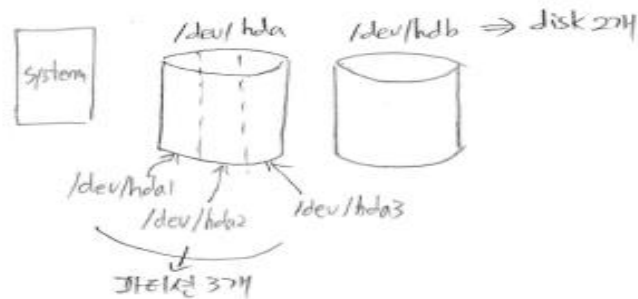
ext2_inode 구조

• Cover Disk = 4T + 4G + 4M + 1024

5. Ext2 파일 시스템

• Ext2의 세부적인 구조와 동작원리를 알아보자.

- IDE 방식 /dev/hd
- SCSI 방식 /dev/sd

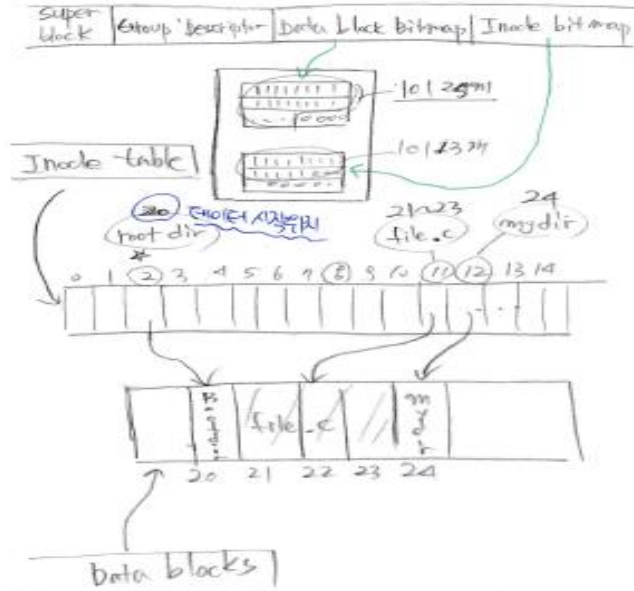
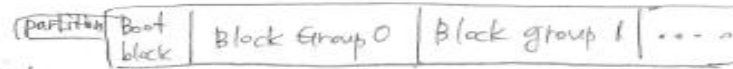


⇒ 디스크가 시스템에 장착되면 디스크는 사용자가 원하는 만큼 파티션으로 분할됨.

단, 각 디스크마다 최대 64개

→ mke2fs = ext2 만드는 명령어

• 파일 시스템은 각 파티션 당 하나씩 만들어짐



1 가상 파일 시스템 (Virtual File System)

1) VFS는 왜 나온 것인가?

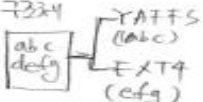
ex) ext4 파일 시스템 NTFS 파일 시스템

dev/hda	dev/hdb
ext4.open	NTFS.open
ext4.read	NTFS.write
ext4.close	NTFS.close

⇒ ext4와 NTFS의 파일 시스템이 제공하는 함수가 달라 굉장히 번거롭다.

그렇기 때문에, 파일 시스템을 고려하지 않는 통합적인 가상 공간이라는 개념이 도입됨.

즉, 미리 관제로 통합해줌.



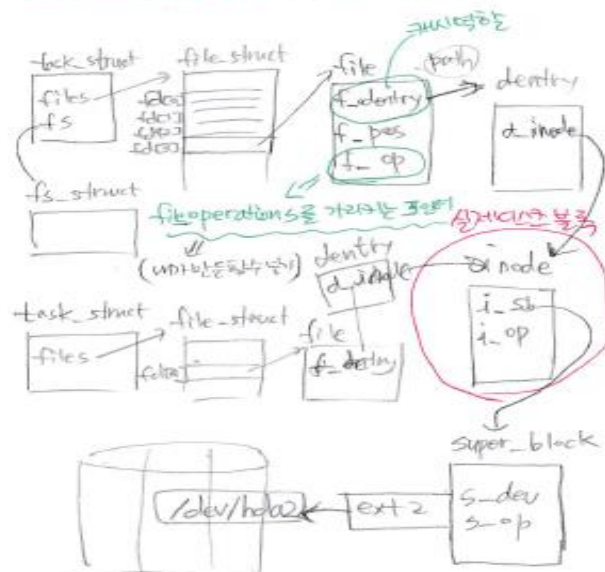
⇒ inode는 파일 시스템을 고려하지 않는다.

ex) Inode_operations로 ext4로 파일 시스템이 정해지면 file_operation도 ext4로 정해진다.

2) VFS는 다양한 파일 시스템과 데이터를 주고 받아야 하는데, 이를 어떻게 관리하나?

- ⇒ ① 슈퍼블록 객체: 현재 사용중인 (mount)된 파일 시스템 하나씩
- ② 아이노드 객체: 특정 파일과 관련된 정보를 담기 위함.
- ③ 파일 객체: 태스크가 open한 파일과 연관되어 있는 정보를 관리. (f_pos)
(두 태스크가 접근하는 위치(offset) 정보는 태스크마다 다르게 유지한다.)
- ④ 디엔트리 객체: 파일의 위치를 캐시로 저장.
- ⑤ path 객체: vfstmount 가상 파일 시스템이 어떤 시스템을 물었는지.

8 태스크 구조 VS VFS 객체



⇒ VFS는 빈 슈퍼블록 객체 만들고 ext2는 파일 시스템 내부 함수로 슈퍼블록을 찾아서 반환하고 sys_open()을 호출하여 빈 inode 객체를 찾아서 반환한다. 다시 채워진 inode는 dentry와 연결시켜 사용자의 task와 연결해준다.

• inode에 i_cdev ⇒ 파일의 속성 (파티션 속성)

i_rdev ⇒ 디바이스 드라이버 주번로

i_mode ⇒ 접근 제어 정보

i_nlink ⇒ inode가 가리키는 파일의 수