File System, Physical Disk

@nenw

at 01.08, 2021 Winter Wheel Seminar

File System Basics

Block Device

하드디스크, SSD 등등...
특정 단위 (섹터)로 I/O할 수 있는 디바이스
버퍼 캐시 사용

디바이스 목록

```
$ ls -al /dev | grep "^b"
$ lsblk
```

Block Device

SATA, SCSI, SAS

/dev/sda, /dev/sdb/, ...

IDE

/dev/hda, /dev/hdb, ...

To find device easily

/dev/disk/by-*

Block Device

= 그냥 스토리지 용량만큼 쌓여있는 데이터 더미

여기서 내가 원하는 데이터 (= 파일)를 이름으로 가져오려면?

내게 필요한 데이터..?

File System

파일을 디바이스에 어떻게 쓰고 읽을 건데?
(OS 마다 다름)

FAT

NTFS / ReFS

HFS+ / APFS

ext(2, 3, 4) / XFS / ZFS / btrfs

Partition

저장장치를 여러 개로 쪼개서 쓰기

보통 하나의 파일 시스템은 하나의 파티션 위에

저장장치 (/dev/sda) → (/dev/sda1, /dev/sda2, ...)

쓰는 이유

파일 시스템 하나가 손상이 나도 다른 파일 시스템에는 문제 없음

용도에 따라 다른 파일 시스템 사용

데이터 용, 운영체제 용을 나누어 사용

Block, Sector

섹터

스토리지 장치의 최소 저장 단위

512 바이트 / 4k 바이트 (Advanced Format, 512e도 가능)

```
# fdisk -l
```

블록

파일시스템의 최소 저장 단위

섹터의 정수배

파일시스템에 따라 블록 크기 조절 가능

일반적으로 최대 4k (Pagesize)

inode

inode Block

파일의 메타데이터 (파일명, 파일 크기, 파일 권한 등)

데이터 블록의 위치

개수는 bytes-per-inode ratio에 의해 파일시스템을 만들 때 결정됨

```
$ df -i
```

Data Block

실제 파일 데이터

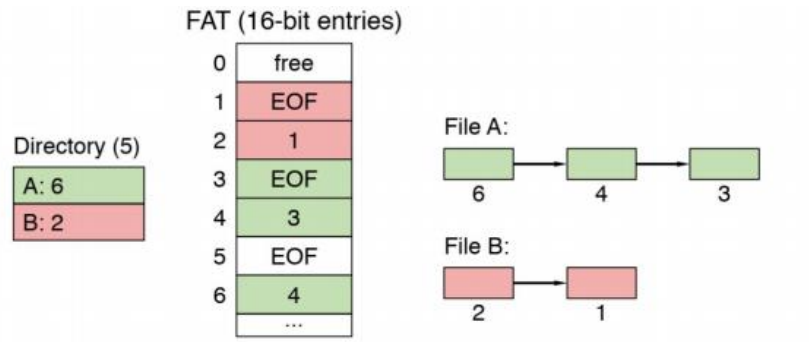
Indexing

Linked Files

파일 메타데이터는 첫 블록을 가리킴

파일은 블록들의 Linked List

예시) FAT



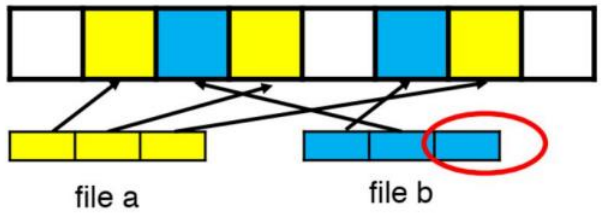
Indexed Files

파일 메타데이터는 모든 블록의 위치를 배열로 가지고 있음

랜덤 액세스가 빠름

배열의 최대 크기로 인한 파일의 최대 크기

(참고로 Linked Files로 소개된 FAT32는 파일 크기가 32비트라 최대 크기가 4GB임)

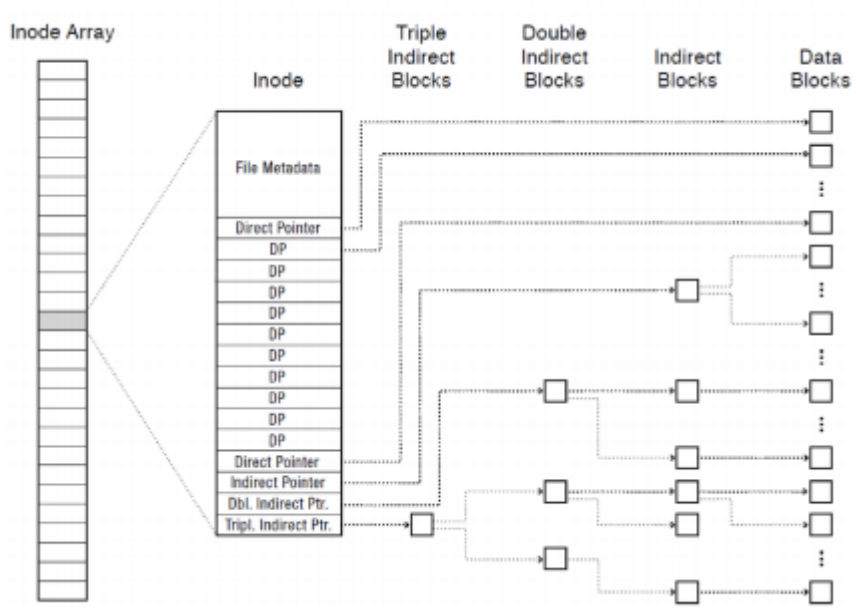


Indexing

Indexed Files

배열의 최대 크기로 인한 파일의 최대 크기

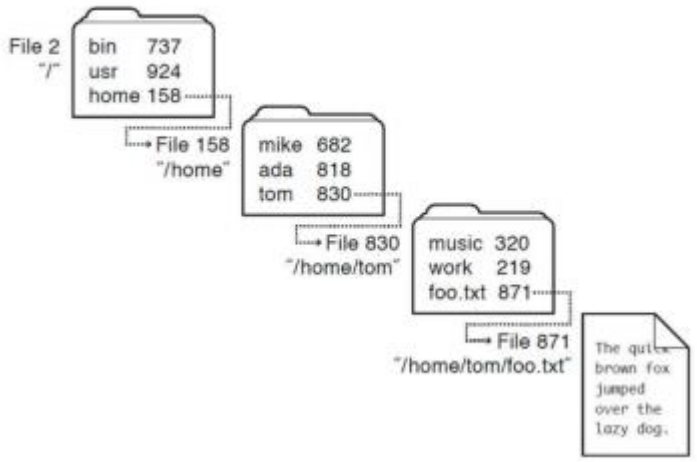
→ Multi-level로 해결



Directory

Directories are files

파일이름과 inode를 매핑하는 파일



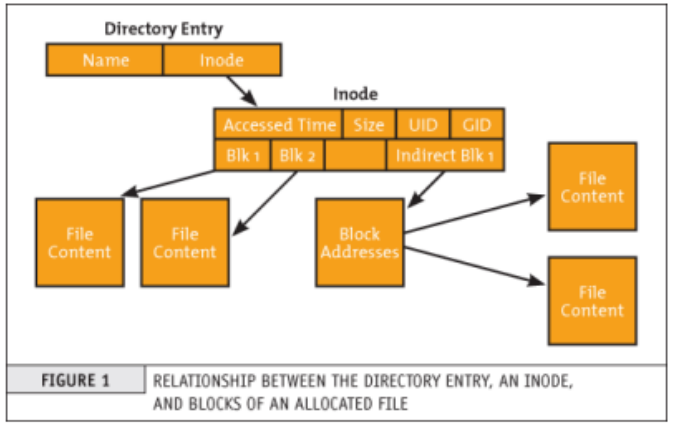


FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

File System Types

ext2/3/4

리눅스에서 기본적으로 많이 쓰이는 파일 시스템

Journaling (ext3부터)

Reserved Blocks

(기본) 5%를 슈퍼유저를 위해 놔둠

파일시스템이 다 차서 명령어가 실패해도 수습할 수 있게

(요즘 하드 크기가 워낙 커져서 너무 많이 할당되는 바람에 낭비되고 있음)

ext / Journaling

파일을 쓰다가 중간에 전원이 꺼지면?

파일의 일부는 최신버전, 나머지 일부는 옛날버전 (Inconsistency)

Journaling

실제 데이터를 쓰기 전에 먼저 변경사항을 기록하기

ext / Journaling

마운트 옵션 "data=모드"

Journaling

Metadata와 Data를 전부 저널에 쓰고, 이후에 파일 시스템에 씀

가장 신뢰성 높음, 두 번 써서 느림

Ordered (기본)

Metadata만 저널에 씀. 이후에 파일 시스템에 씀

Write-back

Metadata만 저널에 씀. 동시에 파일 시스템에 씀

가장 신뢰성 낮음, 가장 빠름

ext4

Allocate-on-flush

직접 쓰기 전까지 할당을 미룸

→ 한번에 여러 블록 할당해 fragmentation 감소

Extents

inode에 구간 내 모든 데이터블록을 나열해 저장하는 대신에 데이터블록의 구간을 저장

→ fragmentation 감소, 저장할 블록 수 감소

하위 디렉토리 개수 제한 완화

사실상 무제한으로 할당 가능

XFS

실리콘 그래픽스에 의해 1990년대 제작됨
고성능, 아직도 굉장히 많이 쓰임
특히 커다란 파일에 강함

스토리지 성능이 중요한 거울에서 XFS를 씀
자원을 조금 더 많이 소모함

ZFS

썬 마이크로 시스템즈에서 개발

자원을 많이 사용함

수많은 기능을 지원

- 캐시 (ARC)
- 자체 RAID 지원 (추후 설명)
- Copy-on-Write
- 스냅샷 지원
- Silent Corruption 방지
- 등등...

리눅스 지원은...

- ZFS on Linux...?

btrfs

ZFS에 대한 리눅스의 대항마
btrfs 또한 기능이 매우 많음

안정성을 원한다면 다른 파일시스템을 쓰는게 나음

NFS

네트워크 파일 시스템

저장공간이 거의 없는 서버 A에서 고용량 서버 B의 파일 시스템을 가져와 쓰기

- B에서 공유할 디렉토리 설정
- A에서 B서버의 IP 주소를 통해 마운트

RPC 기반

- 임의의 포트가 할당되기 때문에 A에서 해당 범위의 방화벽을 B에게 열어줘야함

Swap

디스크의 일부분을 메모리처럼 사용하기

사용용도

서버가 죽는 것 < 서버가 느리게라도 살아있는 것

Hibernation (흔히들 최대 절전모드라고 부르는 그것)

메모리 사용 최적화

리눅스에서는 남는 메모리를 캐시용도로 사용함

정말 쓰잘데기 없는 메모리를 스왑으로 옮기고 캐시를 늘리기

스왑파일 vs 스왑 파티션

스왑파일: 늘리고 줄이고 삭제하고 만들고 등이 용이

스왑파티션: 연속된 블록으로 만들어져 바로 장치에 읽고 쓰기 (속도 빠름)

Swap

디스크의 일부분을 메모리처럼 사용하기

스왑 파일 만들기

```
$ dd if=/dev/zero of=(파일 위치) bs=(버퍼 크기) count=(버퍼 개수)
```

dd: bs만큼 count번 읽어서 if를 of로 복사

명령어

```
$ mkswap (파일 위치) (파일 크기)
$ swapon (파일 / 파티션 위치)
$ swapoff (파일 / 파티션 위치)
$ free
$ swapon -s
```

FUSE

Filesystem in User Space

나만의 파일 시스템 만들기

- 구글드라이브를 파일시스템으로 마운팅? (gdrivefs)
- SSH를 파일 시스템으로 사용? (sshfs)
- Git을 파일시스템으로 사용?? (gitfs)
- 비트토렌트 기반 파일시스템??? (btfs)
- 모든 데이터를 π의 자리수로 저장??????? (pifs)

그 외

FAT

정말 단순함. ESP용으로 쓰임

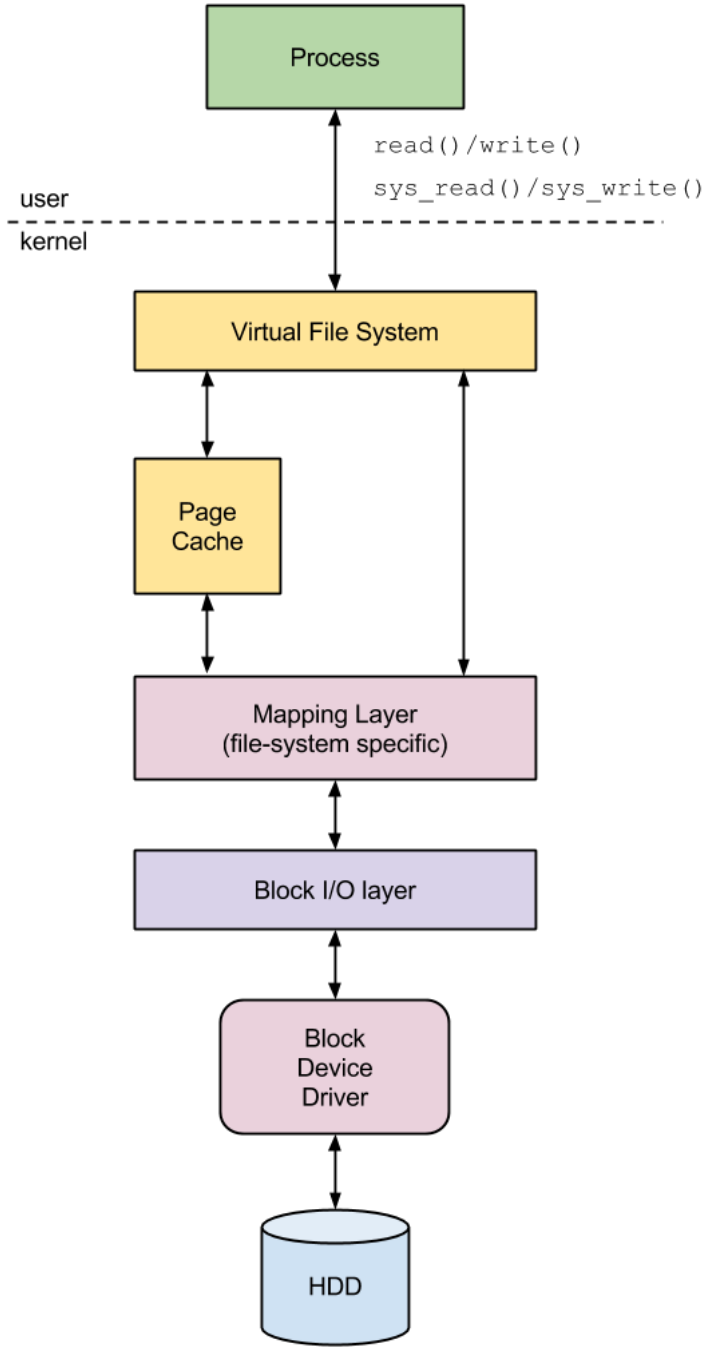
NTFS

윈도우라면 어쩔 수 없이 쓸 파일 시스템.
ntfs-3g로 리눅스에서 사용 가능하나 불안정함

HFS+

그놈의 NFD 정규화

등등...


Process
read()/write()
sys_read()/sys_write()
user
kernel
Virtual File System
Page Cache
Mapping Layer
(file-system specific)
Block I/O layer
Block Device Driver
HDD

File System Tools

MBR, GPT **(From Appendix 1. Booting)**

MBR (Master Boot Record)

하드디스크의 첫 섹터

최대 주 파티션 4개 / 2TB까지만 가능함

부팅에 필요한 코드, 파티션 테이블 등으로 구성

(최대 440 바이트라 사실상 부트로더로 넘겨주기만 함)

GPT (GUID Partition Table)

최대 주 파티션 128개 / 8ZB까지 가능함 (보통 사람들에게 있어선 사실상 무제한)

ESP (EFI System Partition)

Protective MBR

fdisk

대화형 파티션 관리자

w로 쓰기 전까지는 아무런 일도 하지 않음

util-linux 2.23 부터는 GPT도 지원하나 gdisk도 있음

(gdisk: fdisk의 GPT 버전)

파티션 타입

주로 Linux filesystem

스왑은 Linux swap

ESP는 EFI System

`# fdisk (디바이스 파일)`

```
Command (m for help): m

Help:

 DOS (MBR)
  a   toggle a bootable flag
  b   edit nested BSD disklabel
  c   toggle the dos compatibility flag

 Generic
  d   delete a partition
  F   list free unpartitioned space
  l   list known partition types
  n   add a new partition
  p   print the partition table
  t   change a partition type
  v   verify the partition table
  i   print information about a partition

 Misc
  m   print this menu
  u   change display/entry units
  x   extra functionality (experts only)

 Script
  I   load disk layout from sfdisk script file
  O   dump disk layout to sfdisk script file

 Save & Exit
  w   write table to disk and exit
  q   quit without saving changes

 Create a new label
  g   create a new empty GPT partition table
  G   create a new empty SGI (IRIX) partition table
  o   create a new empty DOS partition table
  s   create a new empty Sun partition table
```

parted

얘도 대화형 파티션 관리자

GUI 버전인 gparted도 있음

mkfs

파일 시스템 만들기

mkfs.ext4 등 각 파일 시스템마다 mkfs 명령어가 있음

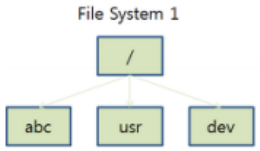
mkfs는 해당 파일을 실행시켜줌

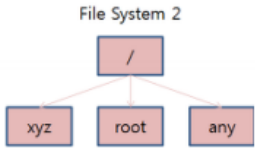
```
$ mkfs -t (파일 시스템) (파티션 장치 파일)
```

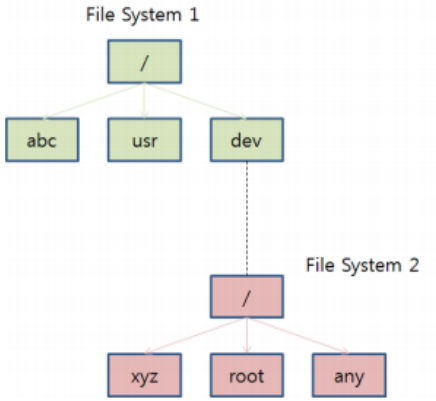
-c 플래그: 배드섹터 검사 (시간이 굉장히 오래 걸림)

그 외에도 많은 옵션이 있음

mount







파일 시스템 마운팅

`$ mount`

현재 마운트 정보

Prettyprint: `$mount | column -t`

`$ mount -t (파일 시스템) -o (옵션) (파티션 장치 파일) (마운트 경로)`

파일시스템 마운트

`$ umount (마운트 경로 / 장치명)`

파일시스템 언마운트

`$ mount -a`

fstab 자동 마운트

fstab

부팅 시에 마운트 할 파일 시스템 목록

/etc/fstab

형식

```
<file system> <mount point> <type> <options> <dump> <pass>
```

파일 시스템 (file system)

마운트 할 파일 시스템 (/dev/sda, UUID=12341234 등)

마운트 포인트 (mount point)

마운트 경로 (/, /boot, /mnt/asdf 등)

타입 (type)

파일시스템 타입 (ext4, xfs 등)

fstab

부팅 시에 마운트 할 파일 시스템 목록

/etc/fstab

형식

```
<file system> <mount point> <type> <options> <dump> <pass>
```

마운트 옵션 (options)

auto: 부팅 시 자동 마운트

rw: 읽기 쓰기 가능 (ro: 읽기만 가능)

exec: 실행파일 실행 가능

nouser: 일반 사용자는 마운트 불가

suid: setuid, setgid 가능

default: rw nouser exec auto suid

그 외 filesystem-specific options도 있음

fstab

부팅 시에 마운트 할 파일 시스템 목록

/etc/fstab

형식

```
<file system> <mount point> <type> <options> <dump> <pass>
```

덤프 (dump)

0: dump 명령어로 백업됨

1: 아님

파일시스템 체크 (pass)

0: 체크하지 않음

1: 루트 파일 시스템 (가장 먼저 체크)

2: 다른 파일 시스템

fsck

파일 시스템 검사

fsck.ext4 등 각 파일 시스템마다 mkfs 명령어가 있음

```
$ fsck -a (파티션 장치 파일)
```

-a 플래그: 자동으로 손상된 블록 수리

-r 플래그: 유저에게 물어보고 수리

꼭 unmount한 상태로 사용!

Physical Disk

SSD

고형 상태 드라이브
주로 WD, 삼성 등등에서 사용

구성요소
메모리 (MLC / TLC / QLC)
디램
컨트롤러

얘도 소모품
낸드 수명 (w/ 온도)
데이터 증발

HDD

하드 디스크 드라이브

주로 WD (+ HGST), Seagate, Toshiba 등에서 제조

구성요소

플래터 (CMR/SMR)

헤드

개복치 **소모품**

배드섹터

스틱션

HDD의 수명?

하드디스크는 언젠가는 사망하게 되어 있음
또한 하드디스크의 수명은 예측하기 어려움
(하루 아침에 사망 가능)

그래도 적어도 예고된 사망은 S.M.A.R.T 를 통해 하드디스크가 보고함
이를 통해 데이터 백업이나 새 하드 구매 등 조치를 취할 수 있음
(참고로 제가 휠에 들어오기 이전부터 사두고 못 쓴 HGST 8TB 하드가 동방에 있습니다.)

S.M.A.R.T

자가 모니터링, 진단 분석, 보고 기술

주요 지표

보류 중인 섹터 수 (Current_Pending_Sector)

재할당 이벤트 수 (Reallocated_Sector_Ct)

회복 불가능 섹터 수 (Offline_Uncorrectable)

읽기 오류율 (Raw_Read_Error_Rate)

등등...

smartmontools

S.M.A.R.T 읽기 및 메일 발송

`$ smartctl -a (디스크 장치명)`

RAID

디스크 여러 개 묶기

디스크를 여러 개 묶어 가용성과 성능을 높이는 것

리빌딩

망가진 디스크를 뺀 후에 새 디스크를 장착

(핫스왑 베이를 통해 그냥 하드만 빼낼 수 있음)

이후 리빌딩 진행해 새로 넣은 디스크에 RAID 내용을 쓰기

RAID

RAID 0

디스크 n개에 읽기 / 쓰기를 분산 (용량 n, 속도 n)

→ 빠른 읽고 쓰기, 하나가 망가지면 전부 망가짐

RAID 1

디스크 n개에 동시에 쓰기 (용량 1, 속도 1)

→ 빠른 읽기, 하나가 망가져도 계속 서비스 가능

RAID

RAID 5

패리티를 이용해 하드 1개까지는 망가지는 걸 살림

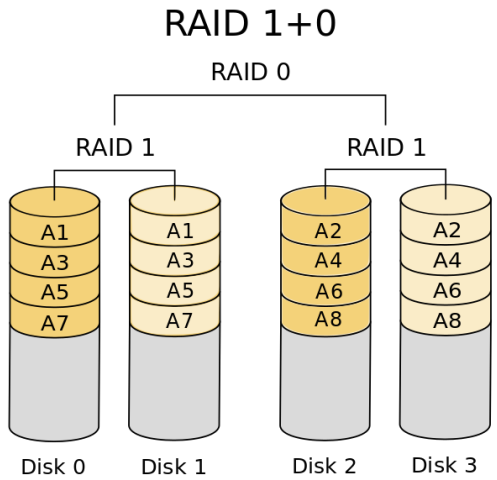
(용량 n - 1)

RAID 6

패리티를 2개 이용해 더 높은 신뢰성

(용량 n - 2)

레이드 둘을 결합 (RAID 10, RAID 60)



RAID

소프트웨어 RAID

무료지만, 하드웨어 RAID보다 성능은 떨어짐

RAID 설정 명령어

`# mdadm`

RAID

하드웨어 RAID

레이드 컨트롤러를 이용해 레이드 구현

현재 미러에서 사용중 (Avago MegaRAID)

RAID 설정 명령어

```
# MegaCli
```

Smartmontools로 S.M.A.R.T 읽기

```
# smartctl -A --device=sat+megaraid,(번호) /dev/sdb
```

Thank You