CS Study

File System - 1

2025. 09. 25

Jiwon Son

목차

시작 전 안내사항

- **설명은 UNIX의 File System에 한정해서만 설명**
  - File System의 종류는 수백가지가 넘으므로, 모두 설명할 수 없음
  - 알아야 하는 용어도 너무 많음
  - 가장 기본이자, 범용성이 높은 시스템에 한정해서 설명

- **얕은 개념 -> 방식 -> 깊은 개념 순서로 설명**
  - 얕은 개념 : 자세한 내용 없이 “이런 게 있다“ 정도로만 넘어갈 것
  - 방식 : 전반적인 동작 흐름
  - 깊은 개념 : “이런 것의 정의와 동작 원리” 에 대해 설명

File

◆ 정의

- 논리적으로 연관된 데이터의 집합

◆ 특성

- File 안에는 단순히 데이터 바이트 스트림만 존재
  - 바이트 스트림 = 2bit 나열
  - 파일에 관련된 모든 메타데이터는 별도 블록에 저장됨
    - 권한, 소유자, 크기, 위치, 타임스탬프 등 : inode에 기록
    - Inode 번호와 파일 이름 : Directory Entry에 기록

◆ File을 관리하기 위해 inode가 필요하고, inode를 활용해 file을 관리하는 시스템이 File System

File System

정의

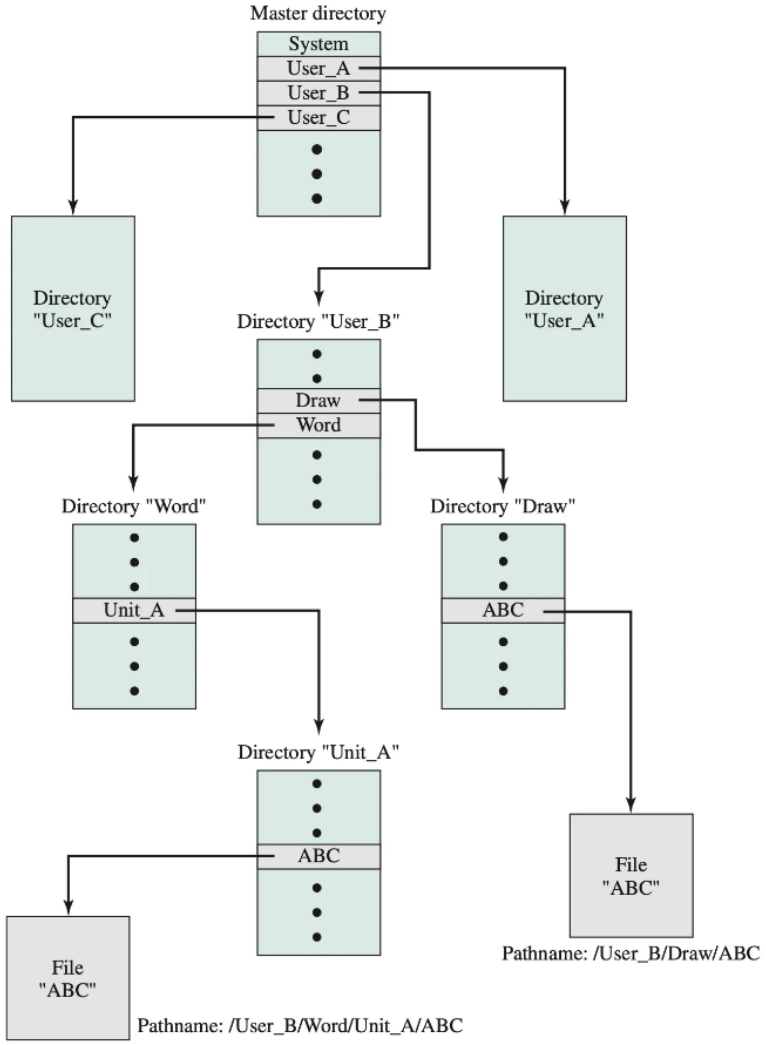
- OS가 데이터를 저장/조작/보호/복구 등 기능을 제공하는 소프트웨어 계층
- =데이터와 메타데이터를 활용해 파일을 관리하는 시스템

동작 방식

- Superblock에 모든 파일 시스템 관련 내용을 저장
- File System(=FS)이 마운트되면 OS는 Superblock을 읽어들임
- 이후 파일에 관련된 모든 동작은 Superblock을 기반으로 동작

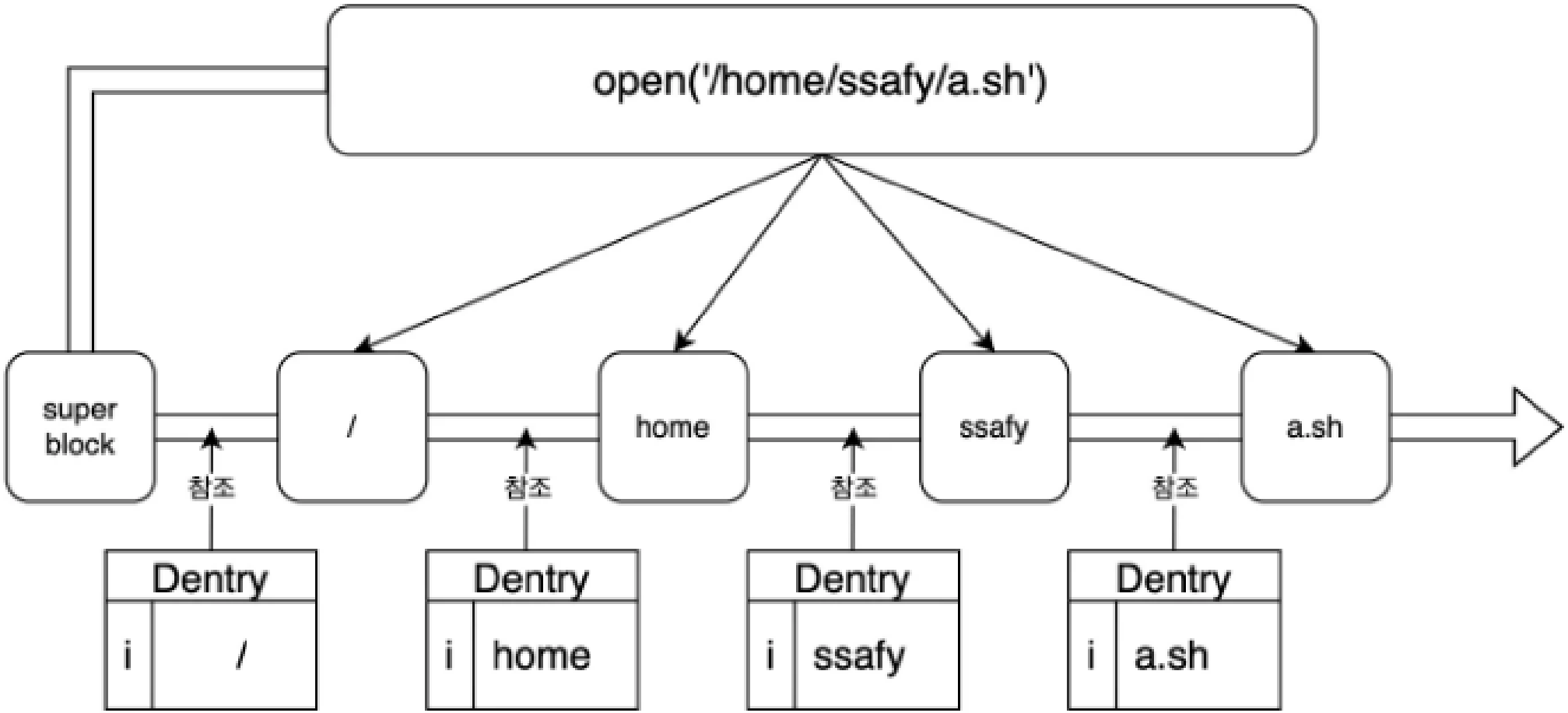
구조

- Tree 형태로 구성됨
  - 현실과 비슷한 방식으로, 직관성 보장
  - 리눅스 기본 철학을 따름
    - “Everything is a file”
    - 단일 namespace에서 다양한 자원을 동일 인터페이스로 사용



File System

◆ 파일 참조 과정



File System

- **한계점?**
  - 리눅스 커널에 기본 탑재된 FS 드라이버만 수십가지
    - 실습에 사용하는 Ubuntu 22.04에만 32개
  - 모든 파일 시스템이 앞서 언급한 내용을 똑같이 해줄것이라 보장 불가
    - 가상화 계층을 만든다면?

- **파일 시스템 가상화**
  - 파일 시스템이 뭐인지는 상관 없이, 동일한 인터페이스로 조작하고 싶음

    => 가상 메모리처럼, 파일관리에도 가상 개념을 도입하자.

```
ssafy@ssafy-VirtualBox:~$ cat /proc/filesystems | wc -l
32
ssafy@ssafy-VirtualBox:~$ lsmod | grep fs
autofs4                57344  2
ssafy@ssafy-VirtualBox:~$ cat /proc/filesystems
nodev   sysfs
nodev   tmpfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cgroup2
nodev   cpuset
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   bpf
nodev   pipefs
nodev   ramfs
nodev   hugetlbfs
nodev   devpts
        ext3
        ext2
        ext4
        squashfs
        vfat
nodev   ecryptfs
        fuseblk
nodev   fuse
nodev   fusectl
nodev   efivarfs
nodev   mqueue
nodev   pstore
nodev   autofs
nodev   binfmt_misc
ssafy@ssafy-VirtualBox:~$ ss
```

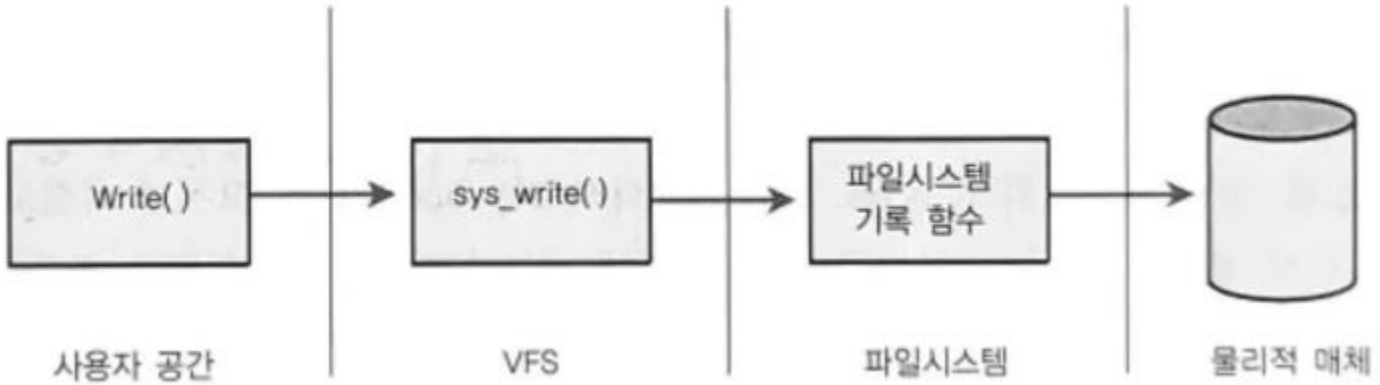
Virtual File System

◆ 정의

- 커널 안에서 여러 종류의 파일 시스템을 추상화해서, 공통 인터페이스로 제공하는 계층
- 사용자는 파일 시스템의 구조/방식/구현 방법에 관계없이 모두 동일 메소드로 사용 가능

◆ 역할

- VFS -> User(=process)
  - POSIX 표준 System call을 동일하게 제공
    - POSIX(Portable Operating System Interface) : UNIX-like OS의 공통 API를 정리한 국제 표준
- VFS -> File System
  - 각 파일 시스템의 구체적인 동작을 파일 연산 함수 포인터 테이블로 연결



Virtual File System

◆ 동작 순서(+ 예시)

- 유저 프로세스가 read(fd, buf, 100); 을 실행했다고 가정.

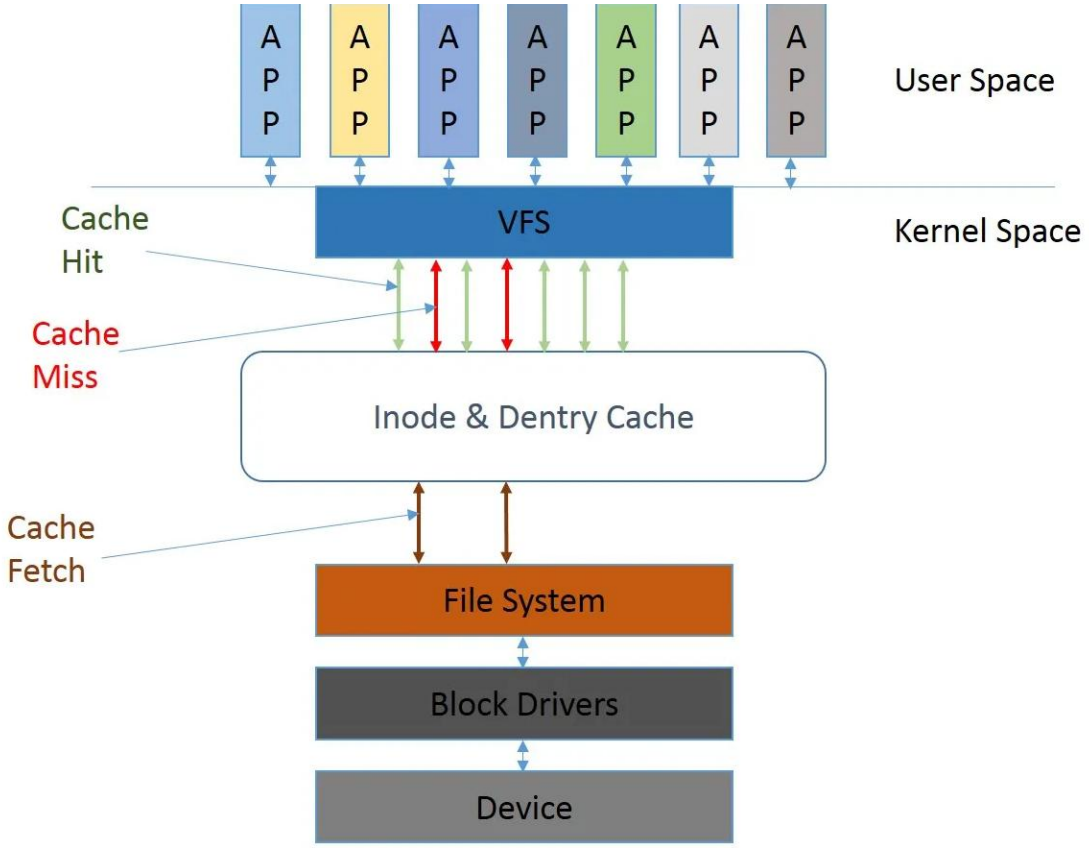
1. File System이 OS에 마운트되면, FS는 자신의 연산 테이블을 미리 VFS에 등록
   Ext4 -> ext4_read()
   NFS -> nfs_read()
   FAT -> fat_read()
2. 파일 연산 명령어가 들어오면, VFS가 경로 탐색으로 inode를 획득
3. 앞서 1.에서 획득한 연산 테이블에 매핑되는 함수 포인터를 호출해서 동작 수행
   VFS 입장에서는 file->f_op->read() 방식으로, 포인터를 따라 해당 FS가 제공하는 코드 호출

◆ 의의

- 사용자 입장에서는 똑같이 read(); 만 호출했지, 내부 구현은 상관 안함
  - 위 예시에서 VFS가 없었다면,
    - ext4라면 드라이버->디스크호출 과정을 알아야 함
    - NFS라면 네트워크를 통해 서버에 요청하고 응답을 기록해야 함
- 사실상 앞서 설명한 흐름 자체가 “가상 파일 시스템 " 이 동작하는 방식

Virtual File System

- 속도 향상에 대한 고찰
  - I/O는 그 자체만으로 엄청 느리다고 그랬는데, 매번 파일을 건드릴 때 마다 이렇게까지 느린걸 감수해야될까?
    - 어떻게든 오버헤드를 줄여서 응답 속도를 올릴 방법이 없을까?
    - > 메모리에 캐시를 3개 배치하자.



Virtual File System

◆ File System Cache

- Inode Cache(=iCache) : 파일 속성 캐시
  - 최근 접근한 inode 구조체를 통째로 캐싱
  - 같은 파일을 여러 번 접근할 때 매번 디스크로 내려가지 않도록 캐싱
  - 메타데이터 작업능률 향상

- Dentry Cache(=dCache) : 경로 캐시
  - 파일 이름 – inode 매핑 정보를 캐싱
  - 매 경로탐색마다 디렉토리 블록을 읽어서 캐싱하는게 오버헤드이므로, 경로 자체를 캐싱하자는 취지
  - 실패도 캐싱함(Negative Cache)
    - 한 시스템이 반복해서 한 파일에 접근하는 경우도 많으므로, “실패도 캐싱 " 해서 에러를 빠르게 뱉도록 지원

- Page Cache(=pCache) : 파일 데이터 캐시
  - 파일의 실제 데이터 블록 내용을 페이지 단위로 캐싱
  - L1캐시<>메모리구조 에서 사용한 로직 그대로, 메모리<>보조기억장치 에 적용
    - 캐시 사용 목적(RW 오버헤드 감소), dirty bit 사용 등, 웬만한 경우에서 목적과 방법이 일치함

Github 질문 목록

◆ File Descriptor와 File System에 대해 설명해주세요.

◆ File Descriptor

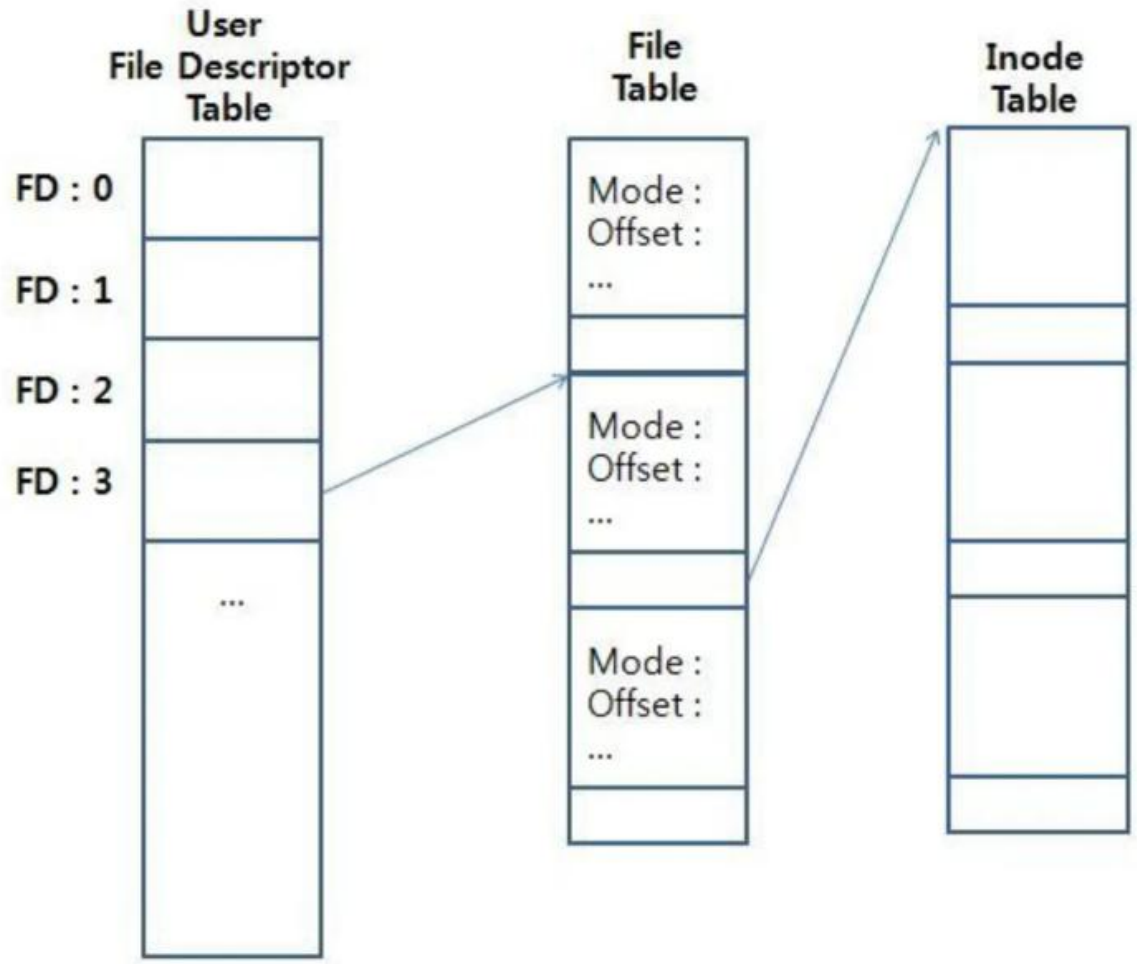
- 정의 : 프로세스가 파일을 열었을 때, 커널이 프로세스에게 반환하는 정수(Integer) 핸들
  - 프로세스 입장에서는 이 번호가 열린 파일을 나타내는 번호
  - 커널 입장에서는 이 번호가 파일 객체를 가리키는 인덱스
- 파일 읽기를 최초로 시도할 경우, 커널이 번호 하나를 같이 리턴
  - 일반적으로 0은 표준입력, 1은 표준출력, 2는 표준에러.
  - 남은 숫자들 중 가장 작은 수 리턴
- 파일 읽기를 다시 시도할 경우, 방금 받았던 번호를 통해 연산을 수행

◆ File System

- 정의 : 저장 장치 단위로 파일/디렉토리 구조를 제공하는 논리적 구조
- Superblock, Inode, Dentry 등을 활용해서 파일 자체를 핸들링하는 방법을 제공

Github 질문 목록

- **File Descriptor 구조, 방식(참고)**
  - Open(“file.txt”, 0_RDONLY) 호출했다고 가정
    - 커널은 해당 파일에 대해 inode, file struct 준비
    - 해당 프로세스의 File Descriptor 테이블에서 빈칸을 찾고, 파일 객체를 가리키도록 연결
    - 위에서 사용한 File Descriptor 인덱스 번호 반환
    - 프로세스는 read(3, buf, size)처럼 FD 번호를 통해 I/O 요청 수행
    - Close(3) 호출시 FD 테이블에서 해제



Github 질문 목록

- **i-Node가 무엇인가요?**
  - 다음 시간에 설명(내용이 너무 많음)

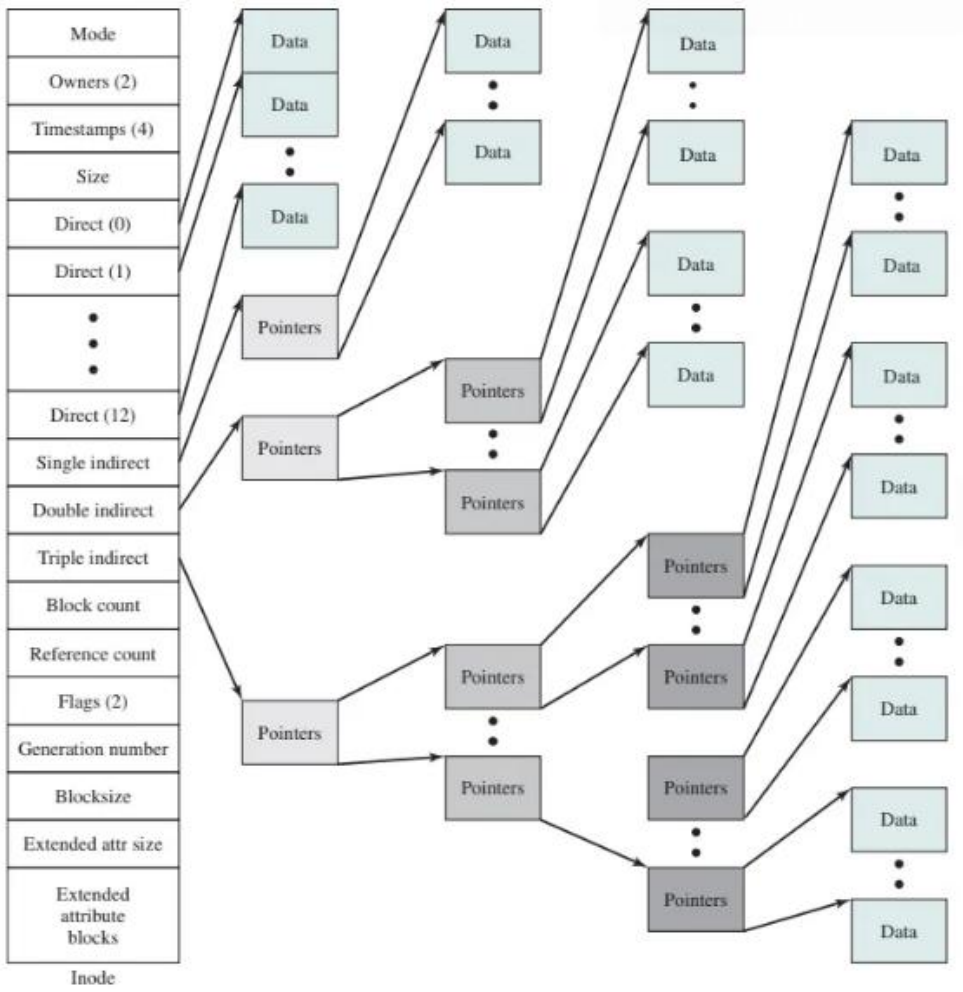


Figure 12.15 Structure of FreeBSD Inode and File

Github 질문 목록

- **프로그래밍 언어 상 제공하는 파일 관련 함수는 파일을 어떤 방식으로 읽어들이나요?**
  - VFS에서 제공하는 POSIX 표준 System call을 호출해서 읽어들임
    - 사용자 프로세스나 개발자는 실제 물리 디스크에서 어떻게 파일을 가져와야할지 몰라도 됨
    - 심지어 파일 시스템이 어떤 FS인지도 몰라도 됨
  - VFS 계층을 통해 추상화된 방식으로 파일 관련 함수를 동작함

EOF