Part 04. 리눅스 커널 입문

# Chapter 04. 파일시스템

## 진행 순서

| | |
|---|---|
| Chapter 04_01 | 파일시스템 개요 |
| Chapter 04_02 | Ext 파일시스템 |
| Chapter 04_03 | 가상 파일시스템 (VFS) |

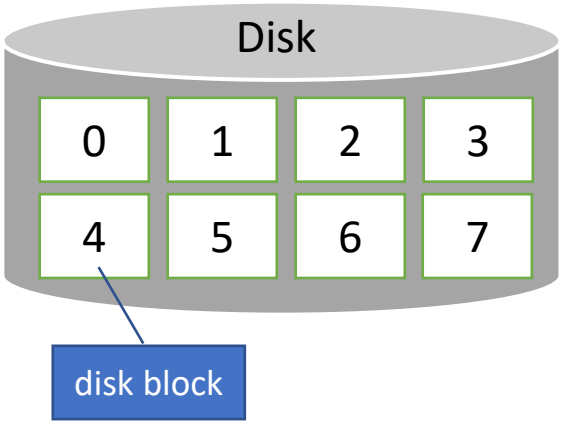
# Chapter 04_01 파일시스템 개요

파일시스템이란?
파일시스템(file system, 문화어: 파일체계)은 컴퓨터에서 파일이나 자료를 쉽게 발견 및 접근할 수 있도록 보관 또는 조직하는 체제를 가리키는 말이다. – Wikipedia

리눅스의 주요 파일 시스템
- 확장 파일 시스템 (ext , ext2 , ext3 , ext4)
- ZFS
- ResierFS
- XFS

디스크 블록
파일시스템은 전체 디스크 저장 공간을 디스크 블록 단위(ex. 4KB) 단위로 나누어 관리한다.
디스크 블록 사이즈가 크면 한번에 읽어 들이거나 쓰는 단위가 커지므로,
실제 물리적인 I/O 접근 횟수가 줄어들어 I/O 병목현상을 줄일 수 있다. 속도가 빠름
단점으로 외부단편화 현상으로 낭비되는 공간이 많아진다.

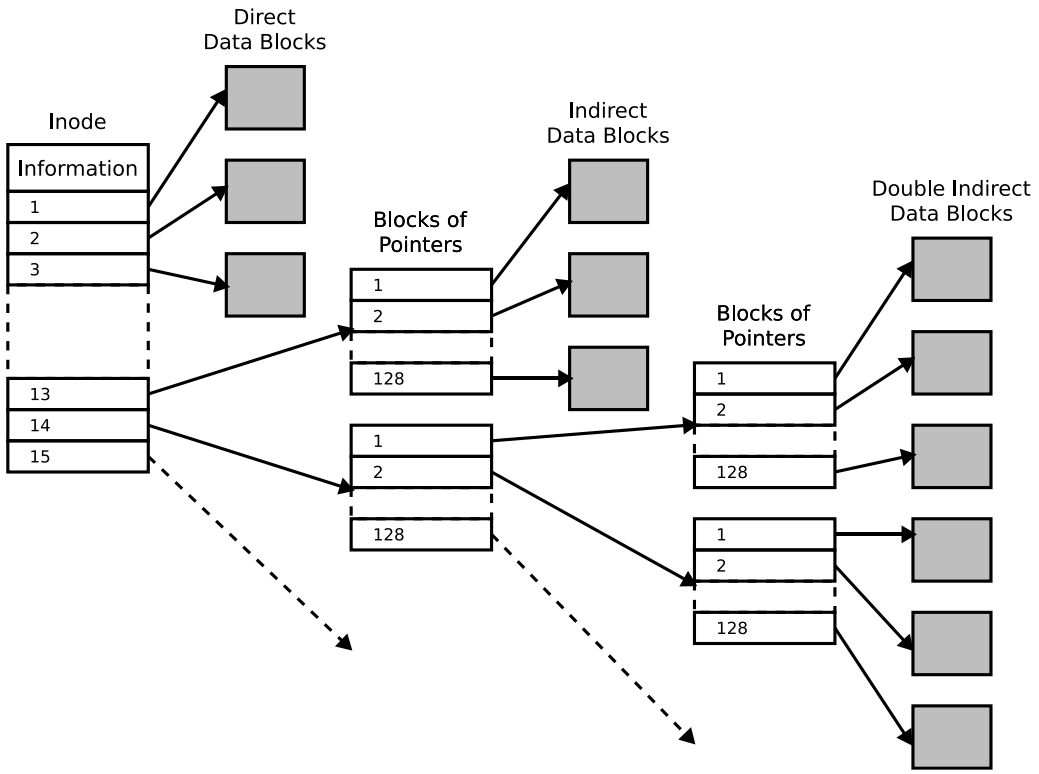
파일시스템은 디스크 저장 공간을 논리적인 디스크 블록들의 집합으로 본다.
디스크 블록은 0, 1, 2 등의 논리적인 번호를 가진다.
이 논리적인 디스크 블록의 번호를 통해 디스크 저장 공간에 접근한다.



# Chapter 04_02 Ext 파일시스템

아이노드 (inode)
모든 파일이나 디렉터리는 아이노드로 표현됩니다.
아이노드는 크기, 권한, 소유권, 그리고 파일이나 디렉터리의 디스크의 위치에 대한 데이터를 포함합니다.
- (/include/linux.fs.h)

```
struct inode {
        umode_t                         i_mode;
        unsigned short                  i_opflags;
        kuid_t                          i_uid;
        kgid_t                          i_gid;
        unsigned int            i_flags;
        const struct inode_operations   *i_op;
        …
        unsigned long                   i_ino;
        …
        dev_t                           i_rdev;
        loff_t                          i_size;
        struct timespec64       i_atime;
        struct timespec64       i_mtime;
        struct timespec64       i_ctime;
        spinlock_t              i_lock;
        unsigned short      i_bytes;
        blkcnt_t                i_blocks;
        …
```




ext2 파일시스템의 inode 테이블 구조

# Chapter 04_02 Ext 파일시스템

디렉토리 엔트리
- inode를 통해 파일의 속성 정보들과 파일에 속한 디스크 블록의 위치를 찾을 수 있다.
- 디렉토리 엔트리를 통하여 파일의 이름과 inode를 연결할 수 있다.
- (/fs/ext4/ext4.h)

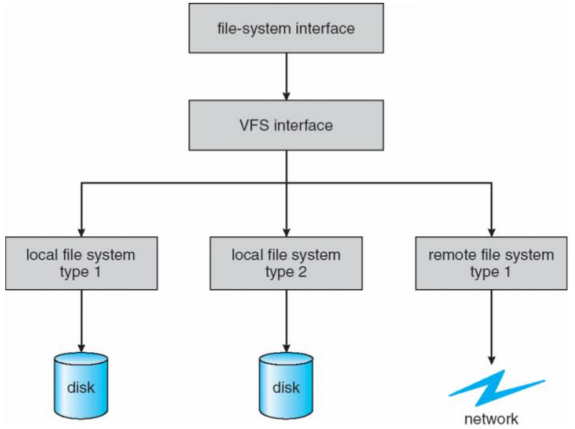
```
struct ext4_dir_entry_2 {
        __le32  inode;                  /* Inode number */
        __le16  rec_len;                /* Directory entry length */
        __u8    name_len;               /* Name length */
        __u8    file_type;              /* See file type macros EXT4_FT_* below */
        char    name[EXT4_NAME_LEN];    /* File name */
};

#define EXT4_FT_UNKNOWN         0
#define EXT4_FT_REG_FILE        1
#define EXT4_FT_DIR             2
#define EXT4_FT_CHRDEV          3
#define EXT4_FT_BLKDEV          4
#define EXT4_FT_FIFO            5
#define EXT4_FT_SOCK            6
#define EXT4_FT_SYMLINK         7
```

# Chapter 04_03 가상 파일시스템 (VFS)

가상 파일시스템?
리눅스 운영체제는 여러 파일시스템을 지원한다. (ext2, ext3, ext4, ZFS, RasierFS, XFS, msdos, ntfs, …)
시스템 내에 파티션 별로 각기 다른 파일시스템을 사용할 때 사용자 태스크에서
파일 열기, 읽기, 쓰기, 닫기 와 같은 행동을 수행한다고 가정하면,
그럼 사용자 프로세스, 즉 태스크들은 해당 파일이 어느 파티션에 위치하는지 식별하고,
각 파일시스템에 맞는 시스템 콜을 이용하여 파일시스템에 접근하여야 한다.
각 파일시스템 별 호출 함수를 직접 호출해서 사용한다는 것은 불편한다.
ex. ext2_create(), ext2_link(), ext2_redpage(), …., msdos_create(), msdos_readpage(), ….

그래서 리눅스는 파일시스템과 사용자 태스크 사이에 가상적인(Virtual) 계층을 도입하였다.
사용자 계층에서는 일반화된 open(), read(), write(), close() 등의 단일화된 함수를 통해 파일시스템에 접근할 수 있다.
중간 계층에서는 파일 이름을 확인하고 파일시스템이 어떤 것인지 식별한 후 해당 파일 시스템의
적절한 함수를 호출하여 결과값을 반환해준다..
이것이 VFS(Virtual File System) 개념이다.

**“VFS는 open(), read(), write(), close()와 같은
시스템 호출이 파일시스템이나 물리적 매체의 종류와
상관없이 동작하게 해주는 역할을 한다.”**



# Chapter 04_03 가상 파일시스템 (VFS)

VFS 주요 객체

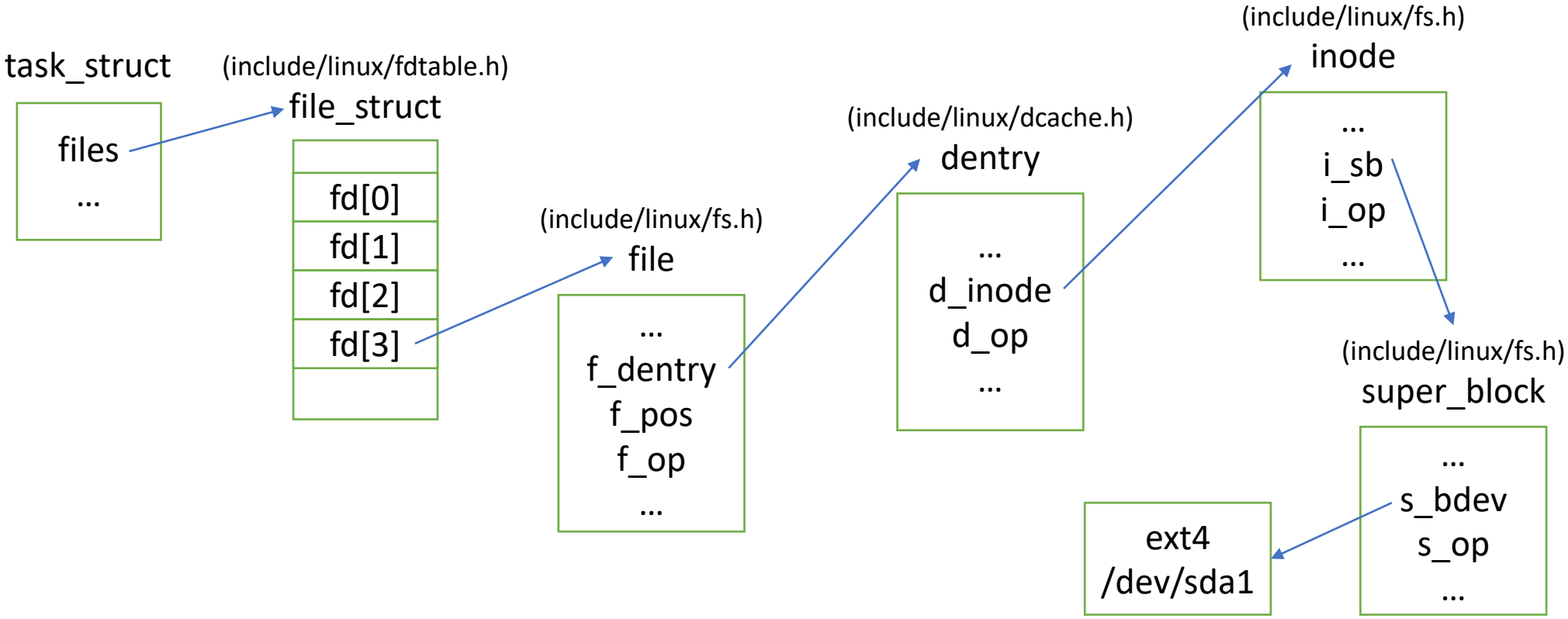
1) 슈퍼블록(superblock) 객체 – 마운트 된 파일시스템을 표현
2) 아이노드(inode) 객체 – 파일을 표현
3) 덴트리(dentry) 객체 – 경로를 구성하는 요소인 디렉토리 항목을 표현
4) 파일(file) 객체 – 프로세스가 사용하는 열린 파일을 표현

상기 객체들에는 동작(operation) 객체가 들어 있다. (커널이 객체에 대해 호출하는 함수)

1) super_operations – write_inode(), sysnc_fs() 등과 같이 특정 파일시스템에 대해 커널이 호출하는 함수
2) inode_oprations – create(), link() 등과 같이 특정 파일에 대해 커널이 호출하는 함수
3) dentry_operations – d_compare(), d_delete() 등과 같이 특정 디렉토리 항목에 대해 커널이 호출하는 함수
4) file_operations – read(), write() 등과 같이 열린 파일에 대해 프로세스가 호출하는 함수

# Chapter 04_03 가상 파일시스템 (VFS)

사용자 프로세스에서 특정 파일을 open() 하는 상황을 가정해보자.
사용자는 파일 이름을 인자로 open() 호출 시 내부적으로 sys_open() 시스템 호출을 요청한다.
VFS는 파일시스템 내부(ex. ext4)의 open 함수를 호출하고 파일시스템은 내부 함수를 이용해,
필요한 정보를 아이노드 객체에 채워서 리턴한다.
VFS는 아이노드 객체를 덴트리 객체에 연결시켜서 사용자의 태스크 구조와 연결해준다.



# Chapter 04_03 가상 파일시스템 (VFS)

그럼 앞에서 설명한 open() 함수를 호출하는 경우 operation은 어떻게 흘러가는지 알아보자.

사용자 레이어에서 open() 함수 호출 시 시스템 콜 sys_open()이 호출되고
VFS 레이어의 file 구조체의 f_op는 open을 가리킨다. (file->f_op->open)
실제 열려는 파일의 유형에 따라 적합한 파일 연산으로 등록된다.
(fifo_open(), blkdev_open(), chrdev_open(), sock_no_open(), 등)
파일 유형에 맞는 파일 연산을 f_op 변수에 등록하고 나면, file->f_op->open() 함수를 호출한다.

open()외 read(), write(), close() 등과 같은 연산도 동일한 흐름으로 동작.