

File System and VFS

단국대학교

컴퓨터학과

2009

백승재

ibanez1383@dankook.ac.kr

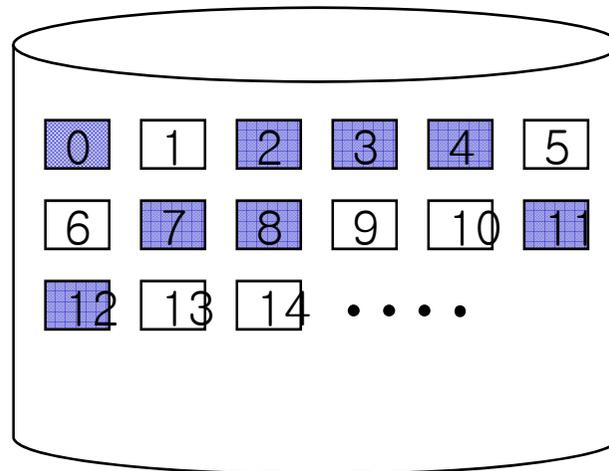
<http://embedded.dankook.ac.kr/~ibanez1383>

강의 목표

- Linux의 FAT과 Ext2 파일 시스템 구조 이해
- VFS의 필요성과 내부구조 파악
- Task와 VFS, 그리고 파일시스템 간의 관계 이해

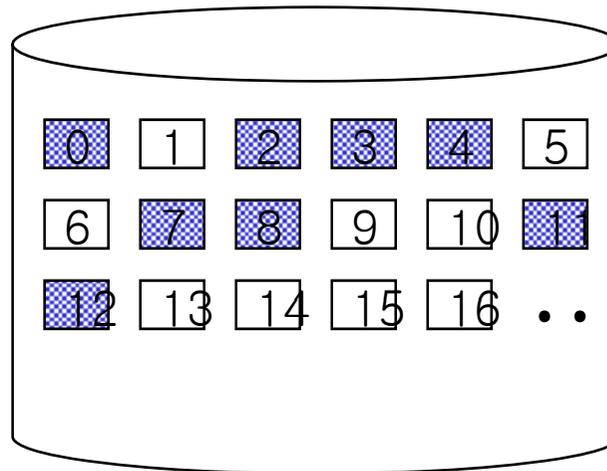
- Bit단위 관리 ? → 논리적인 디스크 블록(4KB)
 - ✓ Block size vs Performance
- 할당 기법
- 어느 블록에 정보를 기입했는가?
 - ✓ 추상적 객체 필요
 - ✓ 파일
- 인덱싱 기법
 - ✓ 블록 체인 기법
 - ✓ 인덱스 블록 기법
 - ✓ FAT
- 성능 향상을 위해
 - ✓ Cylinder group
 - ✓ Fragments
 - ✓ Layout policy
 - 한 Dir내의 파일
 - 한 file내의 block

- 논리적 관점 (커널이 보는 디스크 구조)
 - ✓ 디스크는 **디스크 블록**들의 집합 (disk is a collection of disk blocks)
 - ✓ 디스크 블록의 크기는 보통 페이지 프레임의 크기와 같다 (4K, 8K)



■ 시나리오

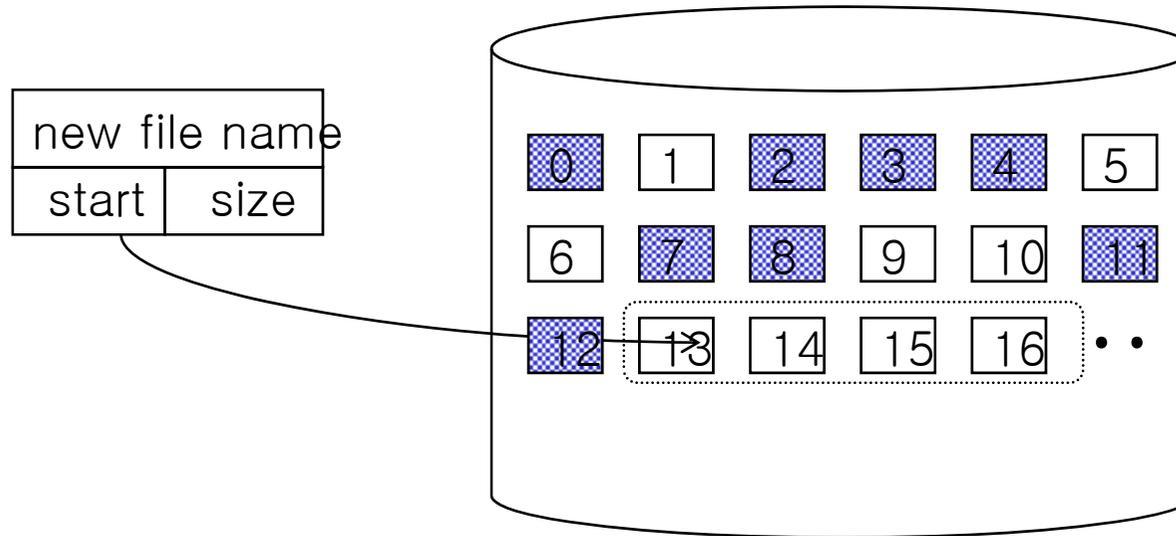
- ✓ 새로운 14 K 크기의 파일을 디스크에게 쓰려고 함
- ✓ 디스크 블록의 크기는 4K로 가정 (따라서 4개의 디스크 블록이 필요)
- ✓ 아래 그림에서 빗금이 있는 디스크 블록들은 이미 사용 중인 (파일의 데이터들을 갖고 있는) 블록이라고 가정
- ✓ 어떤 디스크 블록들을 할당할 것인가? (disk block allocation)



■ 디스크 블록 할당 방법

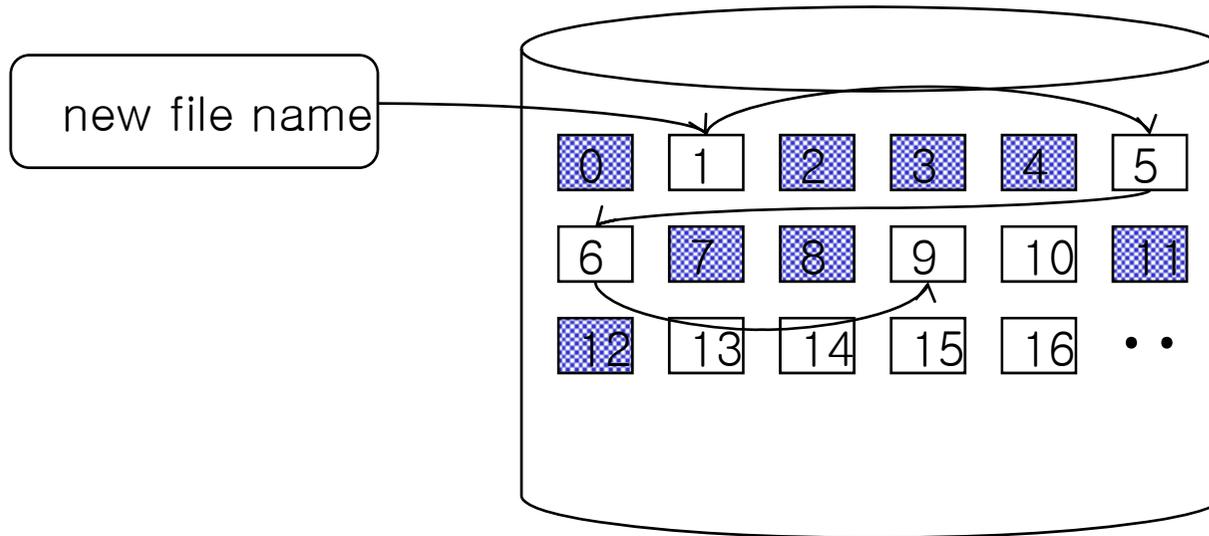
- ✓ 연속 할당 방법 (sequential allocation)
- ✓ 불연속 할당 방법 (non sequential allocation)

■ 연속 할당 방법



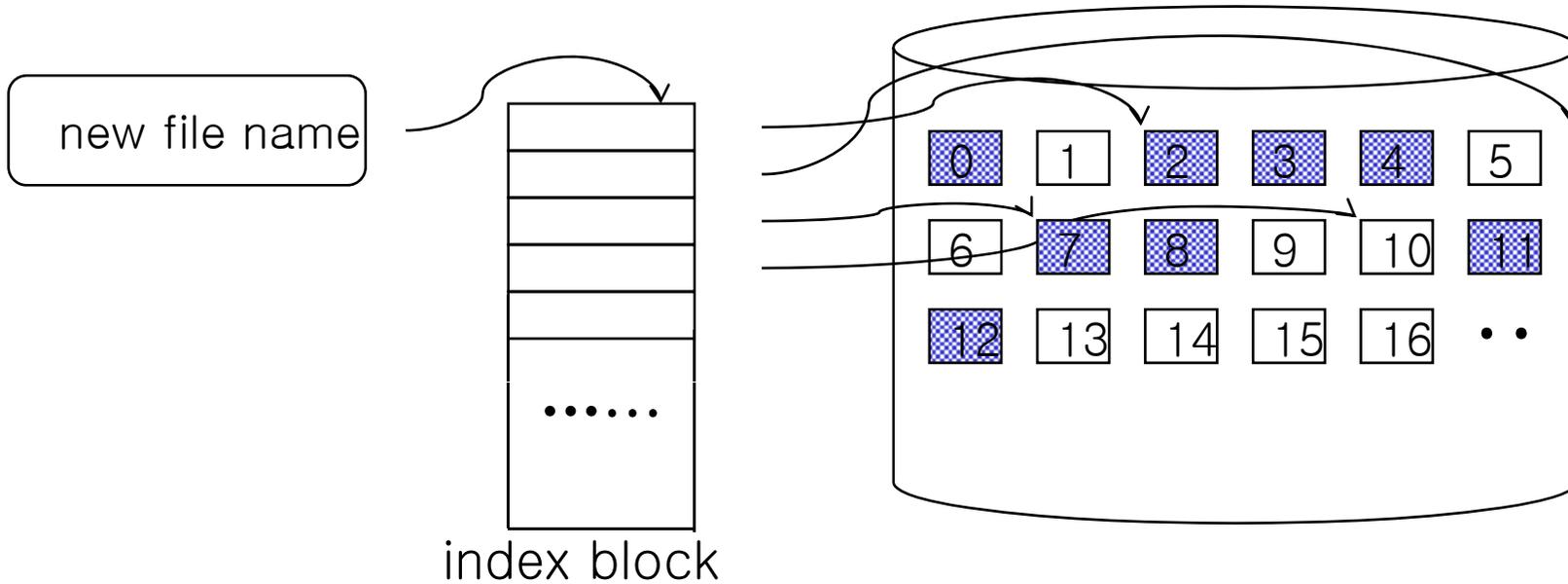
디스크 블록 할당 (3/5)

- 불연속 할당 방법 : 블록 체인 (block chain) 방법



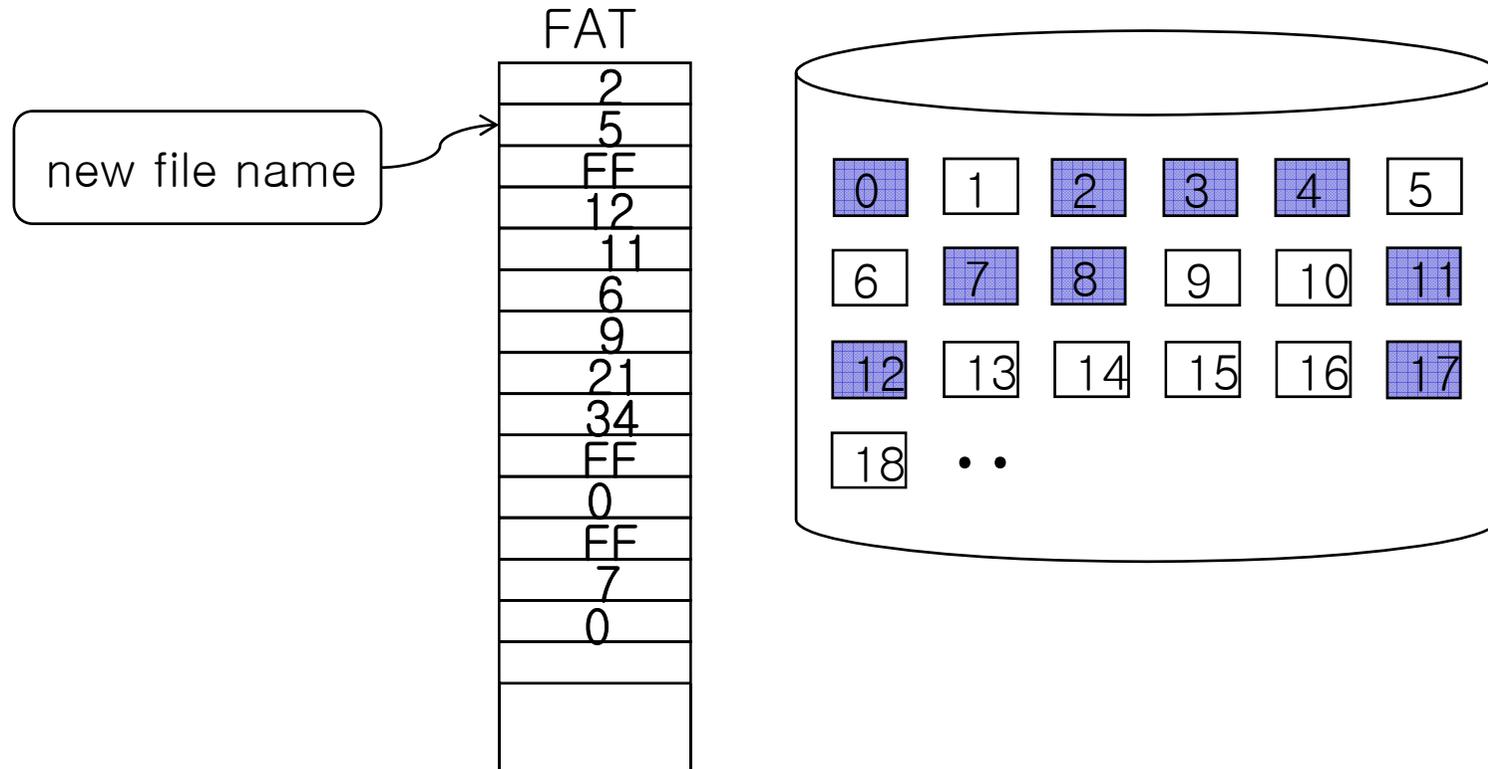
☒ 연속 할당과 불연속 할당 방법 간에 장단점을 논해 보자.

- 불연속 할당 방법 : 인덱스 블록 (index block) 기법



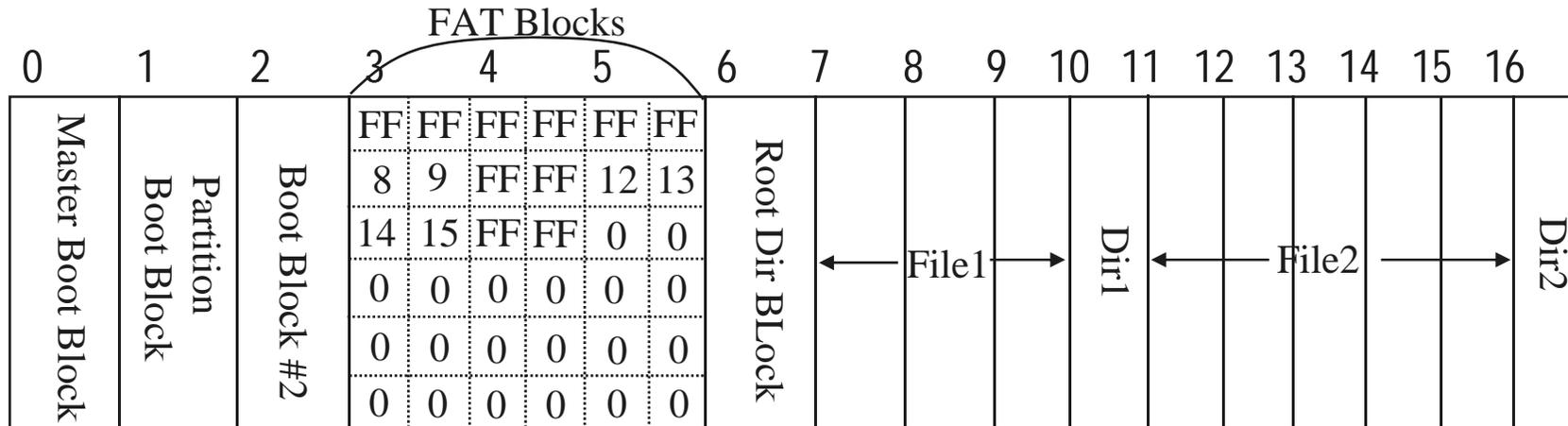
☒ 인덱스 블록이 가득 차면 ?

■ 불연속 할당 방법 : FAT (File Allocation Table)



☒ 블록 체인, 인덱스 블록, FAT 간에 장단점을 논해 보자. Linux는 어떤 방법을 사용하고 있는가?

FAT internal



Status File Name FirstAddr Size

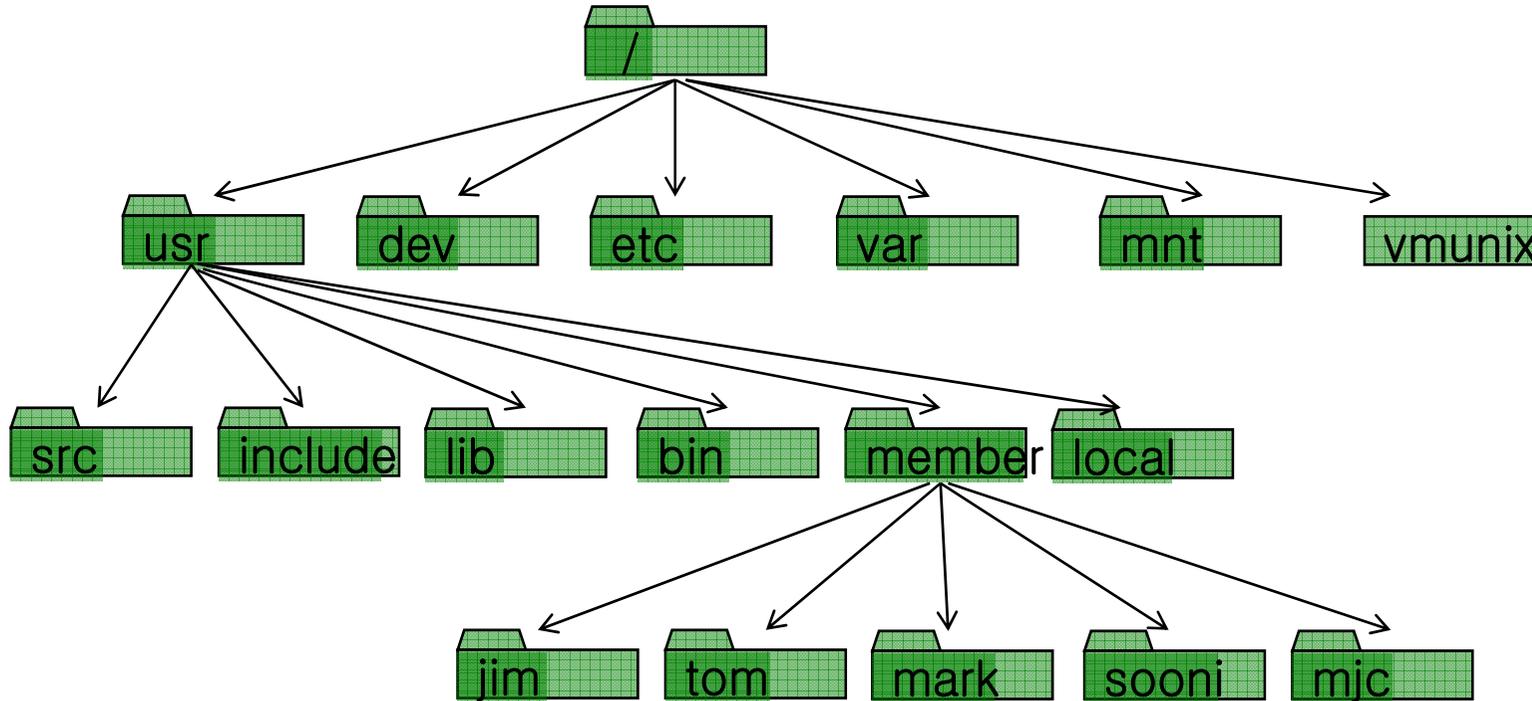
Status	File Name	FirstAddr	Size
F	File1	7	3
D	Dir1	10	1
F	File2	11	5
D	Dir2	16	1

Status File Name FirstAddr Size

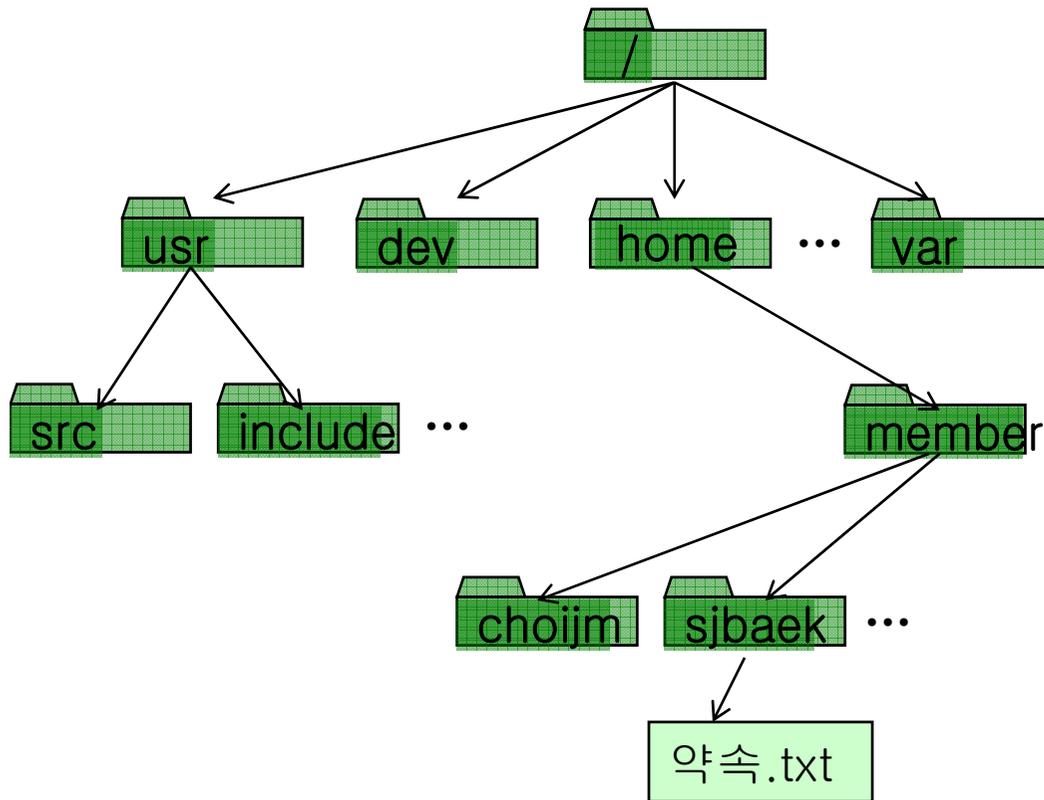
D	.	10	?
D	..	6	?
F	x.hwp	17	5
F	my.doc	22	1

(Source : Dr. dhlee)

- 파일 계층 구조 (hierarchical structure)



디렉토리 계층구조와 메타데이터의 예(FAT기준)



Status File Name FirstAddr Size

6	D	usr	7	3
	D	dev	10	1
	D	home	11	5
	D	var	16	11
		...		

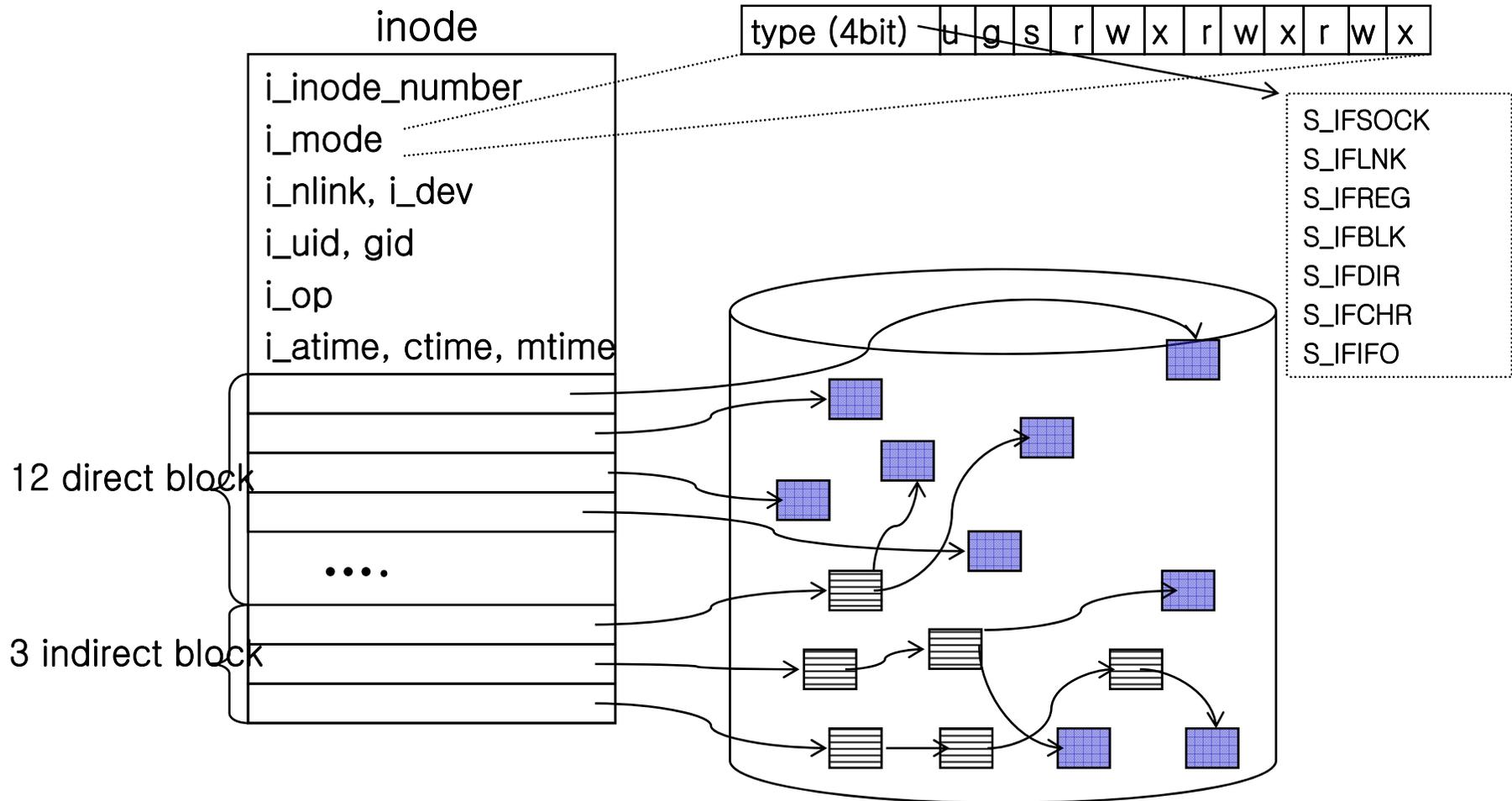
11	D	member	20	2
		...		

20	D	choijm	39	3
	D	sjbaek	43	2
		...		

43	F	약속.txt	48	1
		...		

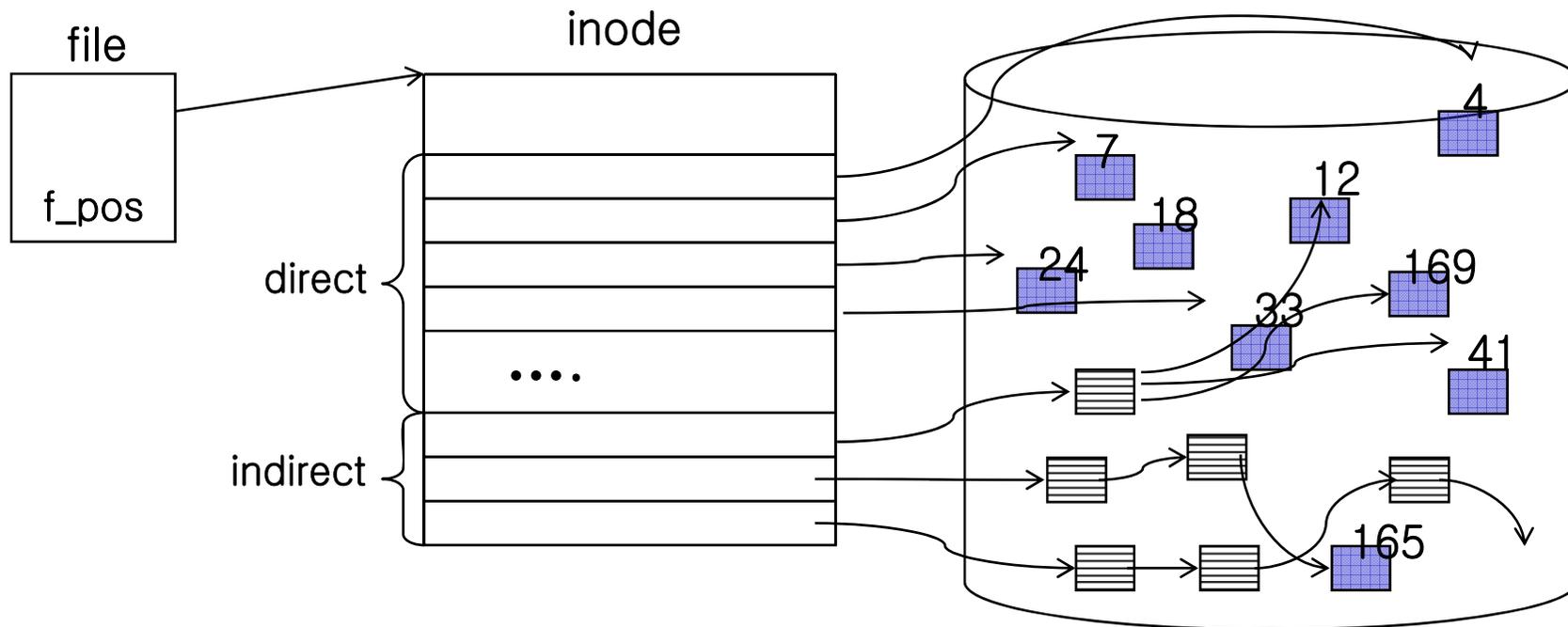
inode 구조 1

- inode `/* include/linux/fs.h, ext2_fs_i.h */`



■ 시나리오

- ✓ 디스크 블록의 크기가 4K로 가정
- ✓ 파일 오프셋(offset)이 10000 이면 어떤 디스크 블록에 접근하게 되는가?
또한 파일 오프셋이 58000 이라면?



■ 디렉터리

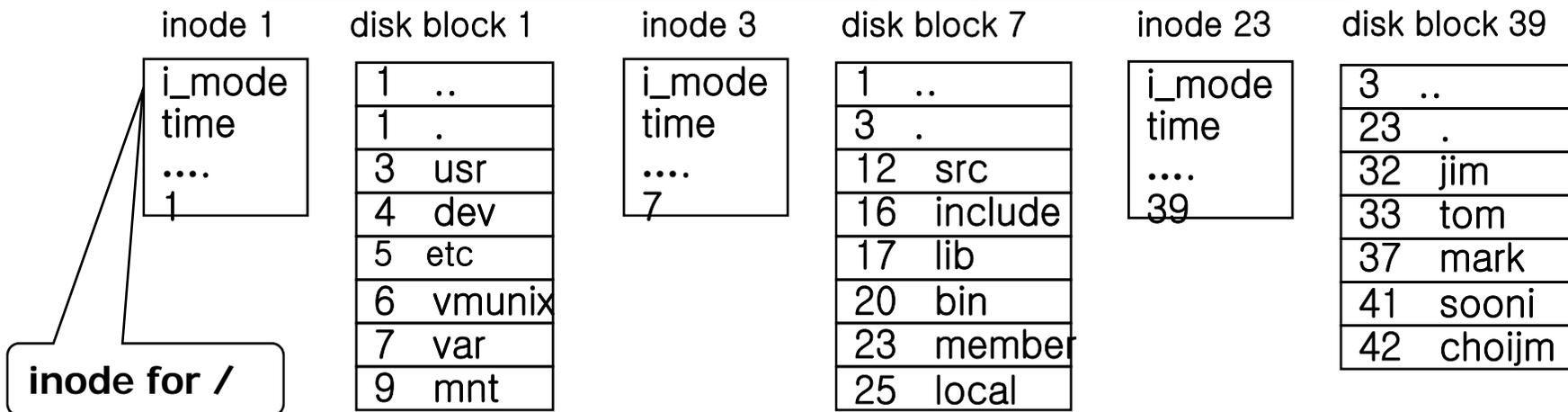
- ✓ 파일 이름과 inode를 연결하는 객체 (namei())
- ✓ 디렉터리 자체도 파일임
- ✓ 계층 구조 제공

Linux의 디렉터리 엔트리

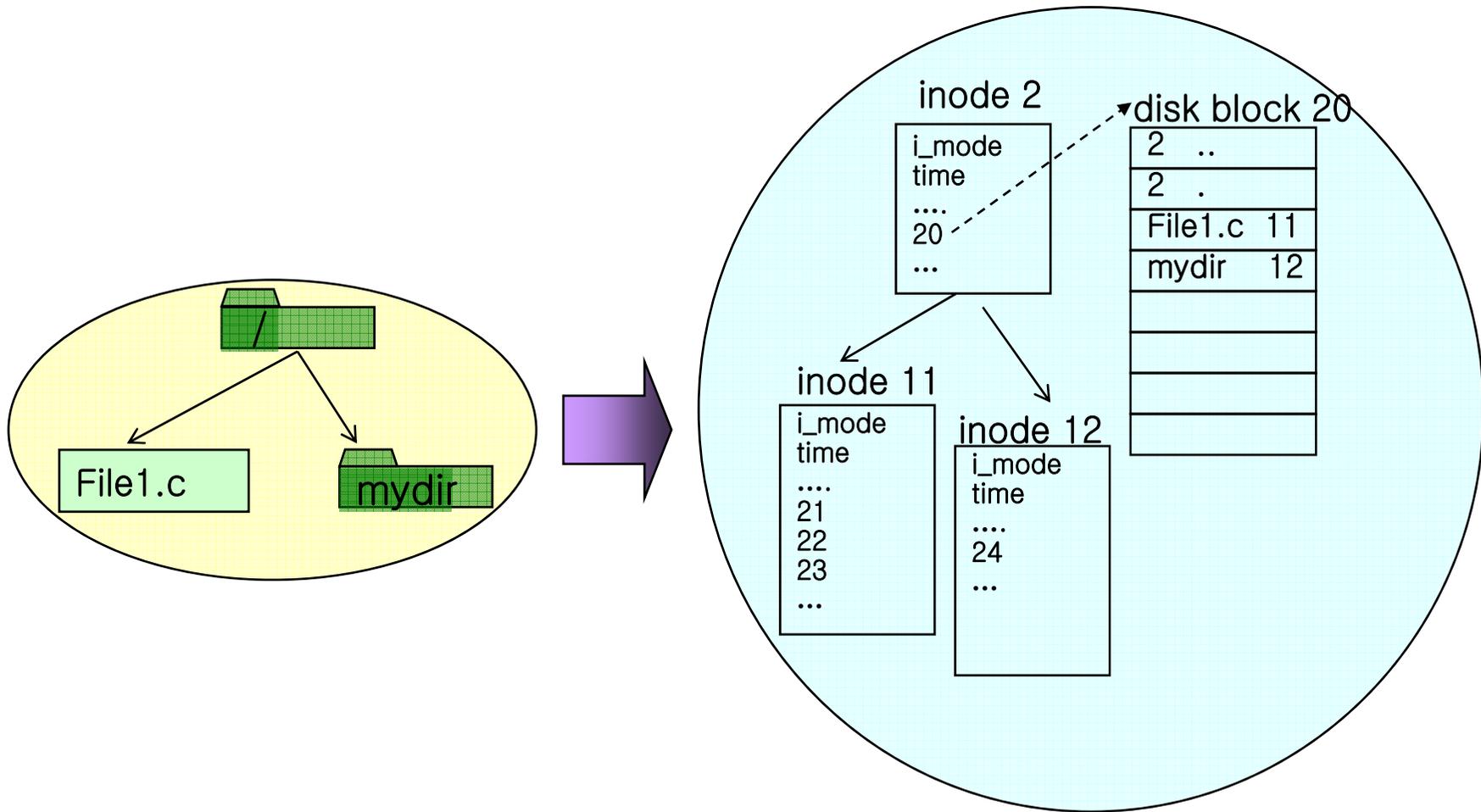
inode number	file name	size	file name
--------------	-----------	------	-----------

directory entry in DOS

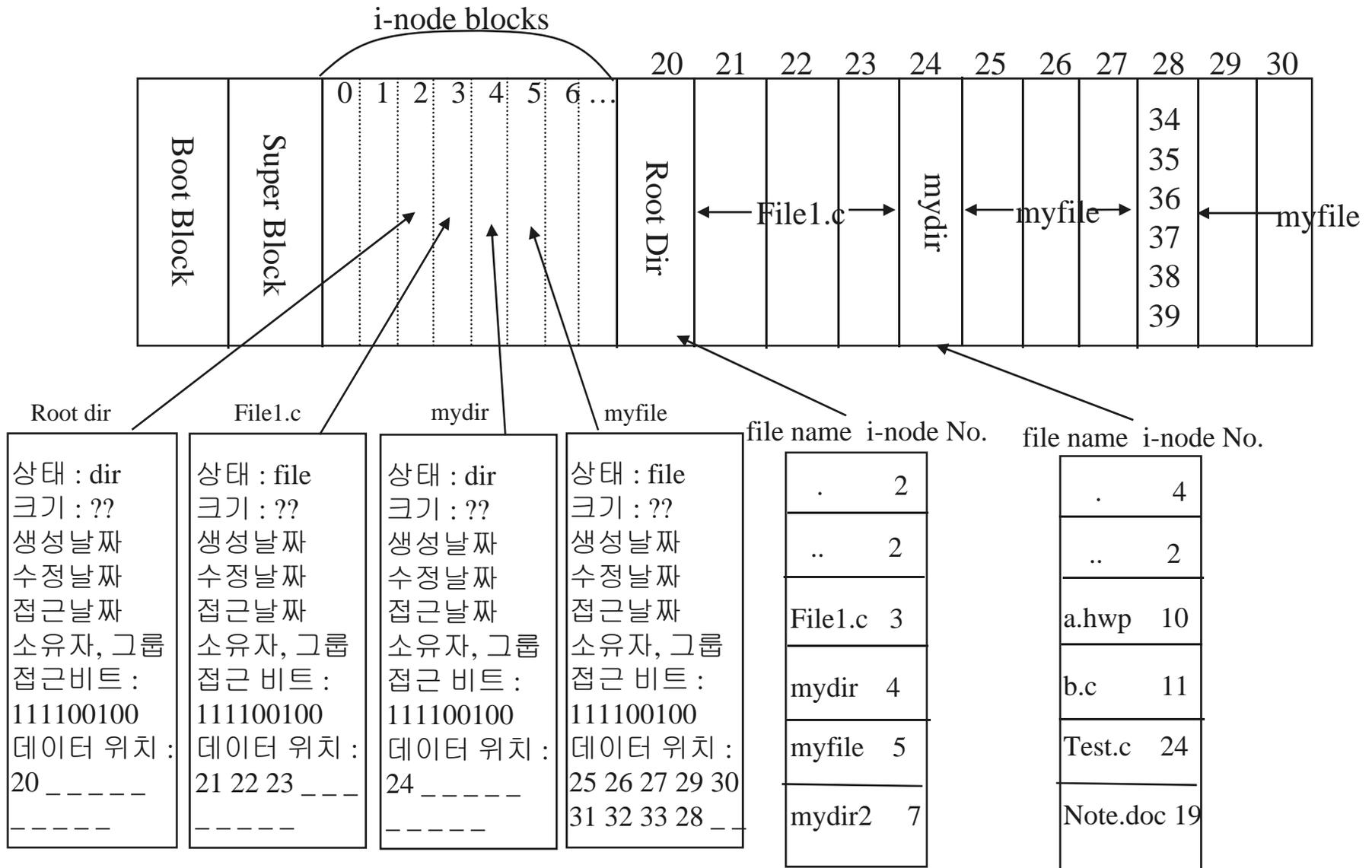
file name	extension	attributes	time	first block number
-----------	-----------	------------	------	--------------------



Ext2의 inode와 파일의 개념적 구조



ext2 internal



(Source : Dr. dhlee)

- 만약 여러분이 FS제작자라면 어떤 구조로 FS를 제작할 것인가?
 - ✓ FAT
 - ✓ Ext2
 - ✓ LFS
 - ✓ ...

Example(2/16)-FS제작

- 만약 여러분이 FS제작자라면 파일을 관리하기 위한 메타 데이터 자료구조를 어떻게 정의할 것인가?
 - ✓ 이름, 크기, 사이즈, 생성시간, data블록의 위치, ...

```

typedef struct {
    unsigned int    blockno;
    unsigned short  offset;
}__attribute__((packed)) ET_T;

typedef struct {
    unsigned char   status;
    time_t         date;
    unsigned long   size;
    char            name[FILE_NAME];
    union{
        ET_T        index[5];
        char         data[30];
    }__attribute__((packed)) uarea;
    unsigned int    next_blkno;
    char            reserved[4];
    char            attr;
}__attribute__((packed)) DIRENTRY_T;

typedef struct {
    unsigned char   status;
    char            reserved[5];
    ET_T            index[9];
    unsigned int    next_blkno;
}__attribute__((packed)) IDXDIRNTRY_T;

```

nebFS

Example(3/16)-FS제작

```

struct ext2_inode {
    __u16    i_mode;    /* File mode */
    __u16    i_uid;    /* Low 16 bits of Owner
    Uid */
    __u32    i_size;    /* Size in bytes */
    __u32    i_atime;   /* Access time */
    __u32    i_ctime;   /* Creation time */
    __u32    i_mtime;   /* Modification time */
    __u32    i_dtime;   /* Deletion Time */
    __u16    i_gid;    /* Low 16 bits of Group
    Id */
    __u16    i_links_count; /* Links count
    */
    __u32    i_blocks; /* Blocks count */
    __u32    i_flags;   /* File flags */
    union {
        struct { __u32 l_i_reserved1; } linux1;
        struct { __u32 h_i_translator; } hurd1;
        struct { __u32 m_i_reserved1; } masix1;
    } osd1; /* OS dependent 1 */
    __u32    i_block[EXT2_N_BLOCKS];
    /* Pointers to blocks */
    __u32    i_generation; /* File version
    (for NFS) */
    __u32    i_file_acl; /* File ACL */
    __u32    i_dir_acl; /* Directory ACL */
    __u32    i_faddr; /* Fragment address */
    .....
    .....
}

```

ext2

Example(4/16)-FS제작

```
struct msdos_dir_entry {
    __s8  name[8],ext[3];    /* name and extension */
    __u8  attr;              /* attribute bits */
    __u8  lcase;            /* Case for base and extension */
    __u8  ctime_ms;         /* Creation time, milliseconds */
    __u16 ctime;            /* Creation time */
    __u16 cdate;            /* Creation date */
    __u16 adate;            /* Last access date */
    __u16          starthi;  /* High 16 bits of cluster in FAT32 */
    __u16 time,date,start;  /* time, date and first cluster */
    __u32 size;             /* file size (in bytes) */
};
```

Msdos FS

```
typedef struct
{
    yaffs_ObjectType type;
    int parentObjectId;
    __u16 sum__NoLongerUsed; // checksum of name. Calc this off the name to prevent inconsistencies
    char name[YAFFS_MAX_NAME_LENGTH + 1];
    __u32 st_mode; // protection // These following apply to directories, files, symlinks - not hard links
#ifdef CONFIG_YAFFS_WINCE
    __u32 notForWinCE[5];
#else
    __u32 st_uid; // user ID of owner
    __u32 st_gid; // group ID of owner
    __u32 st_atime; // time of last access
    __u32 st_mtime; // time of last modification
    __u32 st_ctime; // time of last change
#endif
    int fileSize; // File size applies to files only
    int equivalentObjectId; // Equivalent object id applies to hard links only.
    char alias[YAFFS_MAX_ALIAS_LENGTH + 1]; // Alias is for symlinks only.
    __u32 st_rdev; // device stuff for block and char devices (maj/min)
#ifdef CONFIG_YAFFS_WINCE
    __u32 win_ctime[2];          __u32 win_atime[2]; __u32 win_mtime[2];__u32 roomToGrow[6];
#else
    __u32 rootToGrow[12];
#endif
} yaffs_ObjectHeader;
```

Example(6/16)-FS제작

- 만약 여러분이 FS제작자라면 파일시스템에 어떤 연산을 구현할 것인가? 즉, FS외부로 어떤 interface를 뽑아줄 것인가?
 - ✓ touch → MYFS_file_create()
 - ✓ rm → MYFS_file_delete()
 - ✓ cp → MYFS_file_rename()
 - ✓ cat → MYFS_file_read()
 - ✓ echo >> → MYFS_file_write()
 - ✓ mkdir → MYFS_dir_create()
 - ✓ rmdir → MYFS_dir_delete()
 - ✓ ls → MYFS_dir_read()
 - ✓ ...

Example(7/16)-FS제작

- 다양한 유형의 파일 지원
 - ✓ 정규 파일, 디렉토리, 특수 장치 파일, 링크, 파이프, 소켓, ..

- 파일 시스템 관련 시스템 호출
 - ✓ open, close
 - ✓ read, write
 - ✓ lseek
 - ✓ dup
 - ✓ link
 - ✓ pipe, mkfifo
 - ✓ mkdir, readdir
 - ✓ mknod
 - ✓ stat
 - ✓ mount
 - ✓ sync, fsck

- POSIX interface ?

Example(8/16)-FS제작

```
static struct file_operations neb_dir_operations = {
    readdir:    neb_readdir,
};
static struct inode_operations
neb_dir_inode_operations = {
    create:     neb_create,
    lookup:     neb_lookup,
    unlink:     neb_unlink,
    mkdir:     neb_mkdir,
    rmdir:     neb_rmdir,
    rename:    neb_rename,
    permission: neb_permission,
    symlink:   neb_symlink,
    mknod:     neb_mknod,
};
static struct file_operations neb_file_operations = {
    read:       neb_file_read,
    write:      neb_file_write,
    mmap:       generic_file_mmap,
};
static struct inode_operations
neb_file_inode_operations = {
    lookup:     neb_lookup,
    permission: neb_permission,
    truncate:   neb_truncate,
};
```

nebFS

Example(9/16)-FS제작

```
struct file_operations ext2_dir_operations = {
    read:        generic_read_dir,
    readdir:    ext2_readdir,
    ioctl:      ext2_ioctl,
    fsync:      ext2_sync_file,
};
struct file_operations ext2_file_operations = {
    llseek:     generic_file_llseek,
    read:       generic_file_read,
    write:      generic_file_write,
    ioctl:      ext2_ioctl,
    mmap:      generic_file_mmap,
    open:       generic_file_open,
    release:    ext2_release_file,
    fsync:      ext2_sync_file,
};
static struct super_operations ext2_sops = {
    read_inode:    ext2_read_inode,
    write_inode:   ext2_write_inode,
    put_inode:     ext2_put_inode,
    delete_inode:  ext2_delete_inode,
    put_super:     ext2_put_super,
    write_super:   ext2_write_super,
    statfs:        ext2_statfs,
    remount_fs:    ext2_remount,
};
```

ext2

```
struct inode_operations
msdos_dir_inode_operations = {
    create:  msdos_create,
    lookup:  msdos_lookup,
    unlink:  msdos_unlink,
    mkdir:   msdos_mkdir,
    rmdir:   msdos_rmdir,
    rename:  msdos_rename,
    setattr: fat_notify_change,
};
```

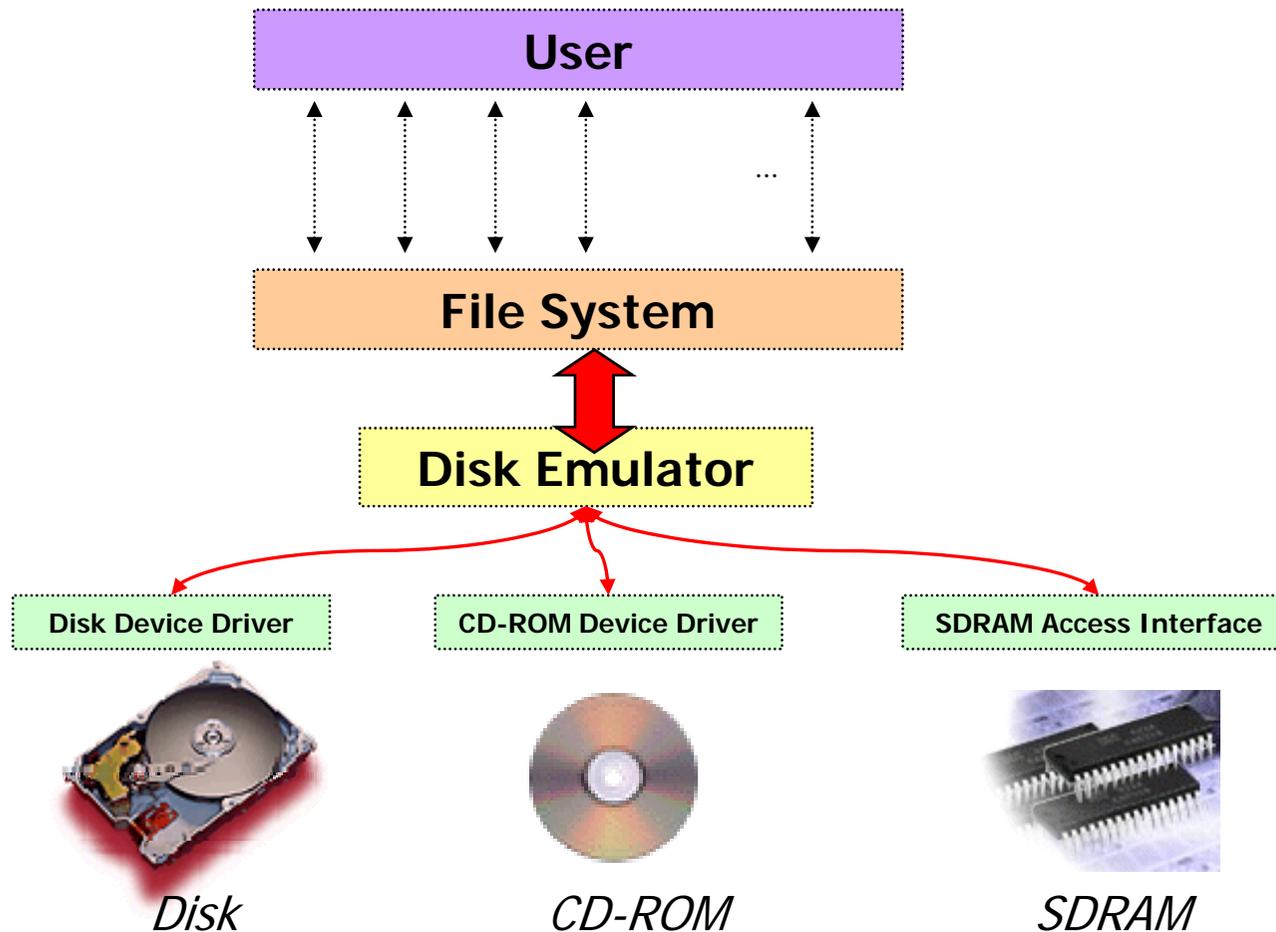
Msdos FS

Example(11/16)-FS제작

```
static struct address_space_operations
yaffs_file_address_operations = {
    readpage: yaffs_readpage,
    prepare_write: yaffs_prepare_write,
    commit_write: yaffs_commit_write
};
static struct file_operations yaffs_file_operations = {
#ifdef CONFIG_YAFFS_USE_GENERIC_RW
    read: generic_file_read,
    write: generic_file_write,
#else
    read: yaffs_file_read,
    write: yaffs_file_write,
#endif
    mmap: generic_file_mmap,
    flush: yaffs_file_flush,
    fsync: yaffs_sync_object,
};
static struct inode_operations
yaffs_file_inode_operations = {
    setattr: yaffs_setattr,
};
struct inode_operations
yaffs_symlink_inode_operations =
{
    readlink: yaffs_readlink,
    follow_link: yaffs_follow_link,
    setattr: yaffs_setattr
};
```

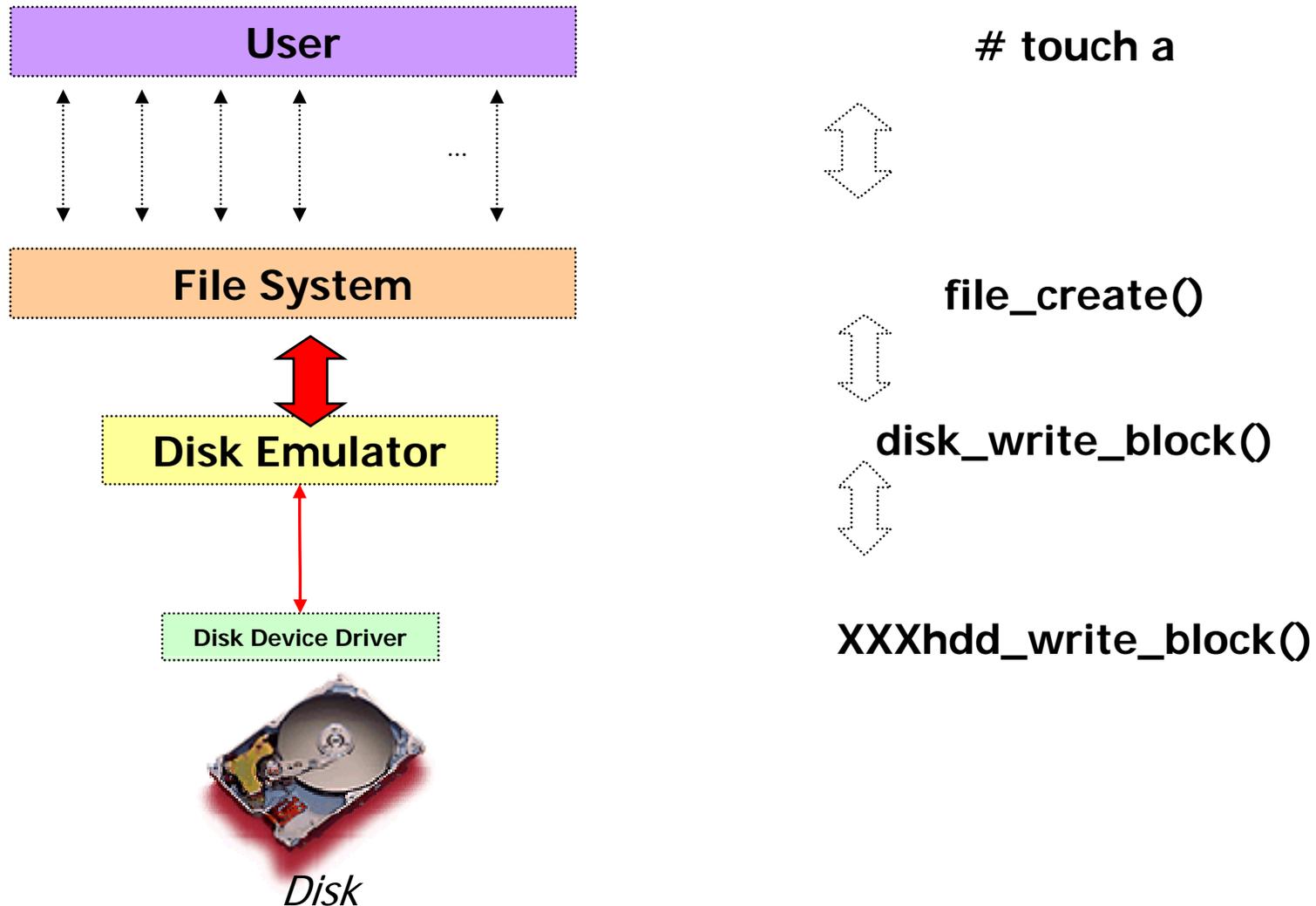
```
static struct inode_operations
yaffs_dir_inode_operations = {
    create: yaffs_create,
    lookup: yaffs_lookup,
    link: yaffs_link,
    unlink: yaffs_unlink,
    symlink: yaffs_symlink,
    mkdir: yaffs_mkdir,
    rmdir: yaffs_unlink,
    mknod: yaffs_mknod,
    rename: yaffs_rename,
    setattr: yaffs_setattr,
};
static struct file_operations yaffs_dir_operations = {
    read: generic_read_dir,
    readdir: yaffs_readdir,
    fsync: yaffs_sync_object,
};
static struct super_operations yaffs_super_ops = {
    statfs: yaffs_statfs,
    read_inode: yaffs_read_inode,
    put_inode: yaffs_put_inode,
    put_super: yaffs_put_super,
    delete_inode: yaffs_delete_inode,
    clear_inode: yaffs_clear_inode,
};
```

- 만약 여러분이 FS제작자라면 만든 FS가 특정 device위에서만 돌도록 만드시겠습니까?



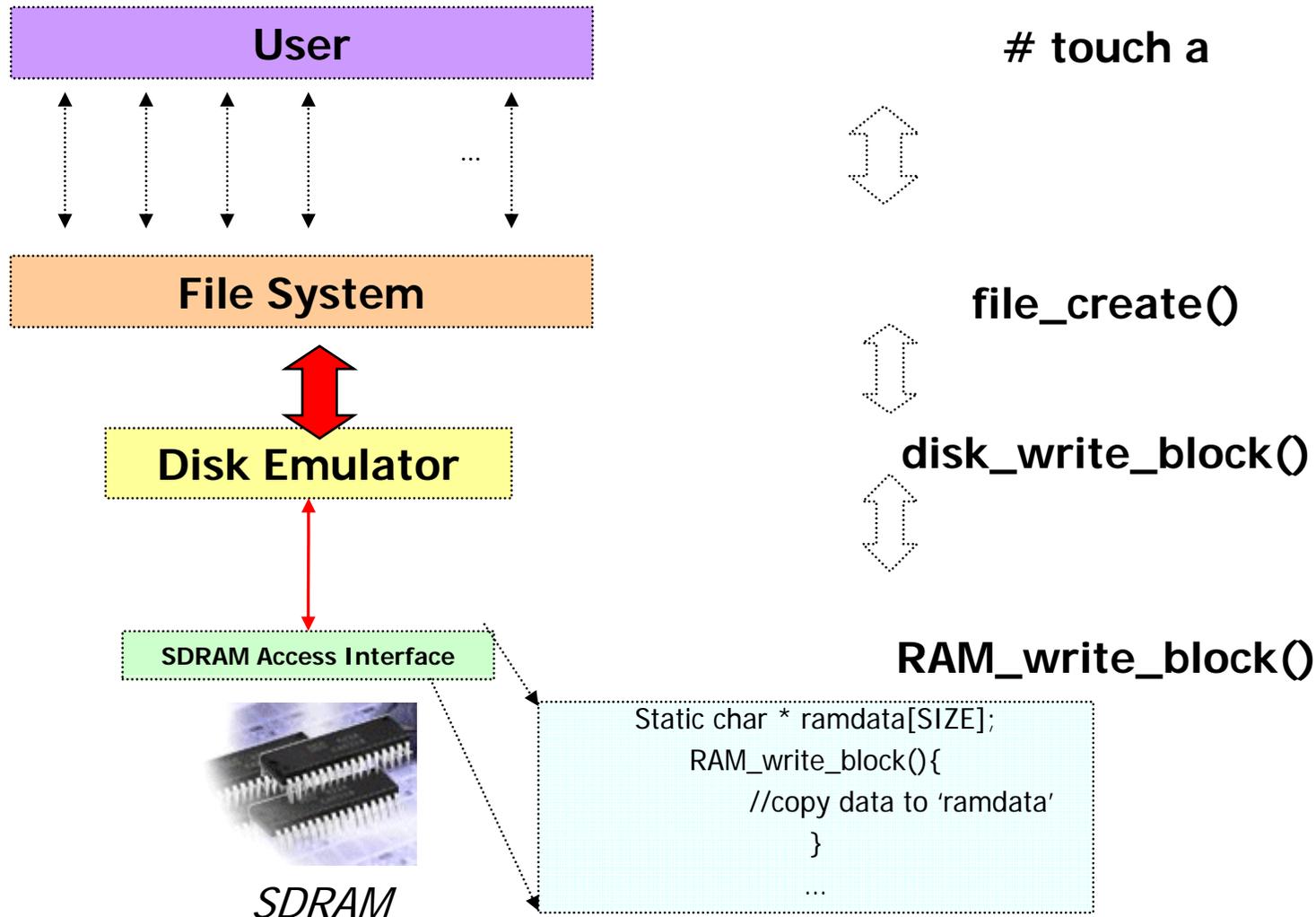
Example(13/16)-FS제작

✓ 만약 Hard Disk 위 에서 돌리고 싶다면?



Example(14/16)-FS제작

✓ 만약 RAM 상에서 돌리고 싶다면?



Example(15/16)-FS제작

■ FS함수 구현의 예(ls)

```
int MYFS_dir_read()
{
    int i, sector, offset, files_in_dir;
    DIRENTRY_T *ep;

    SM_P();
    ep = (DIRENTRY_T *)buf;
    sector = cur_dir;
    offset = 0;

    disk_read_block(sector, buf);
    files_in_dir = ep->size;

    for (i = 0; i < files_in_dir; i++) {
        disk_read_block(sector, buf);
        ep = (DIRENTRY_T *) (buf+offset);
        if (ep->status == DIR) printk(" %14s/ %d", ep->name, ep->size);
        else printk(" %14s %d", ep->name, ep->size); /* ep->status == FILE */
        offset += sizeof(DIRENTRY_T);
        if (offset == 512) {
            sector = FAT[sector];
            offset = 0;
        }
        if (!(i+1) % 4) printk("\n");
    }
    printk("\n");

    SM_V();
    return(FS_SUCCESS);
}
```

Example(16/16)-FS제작

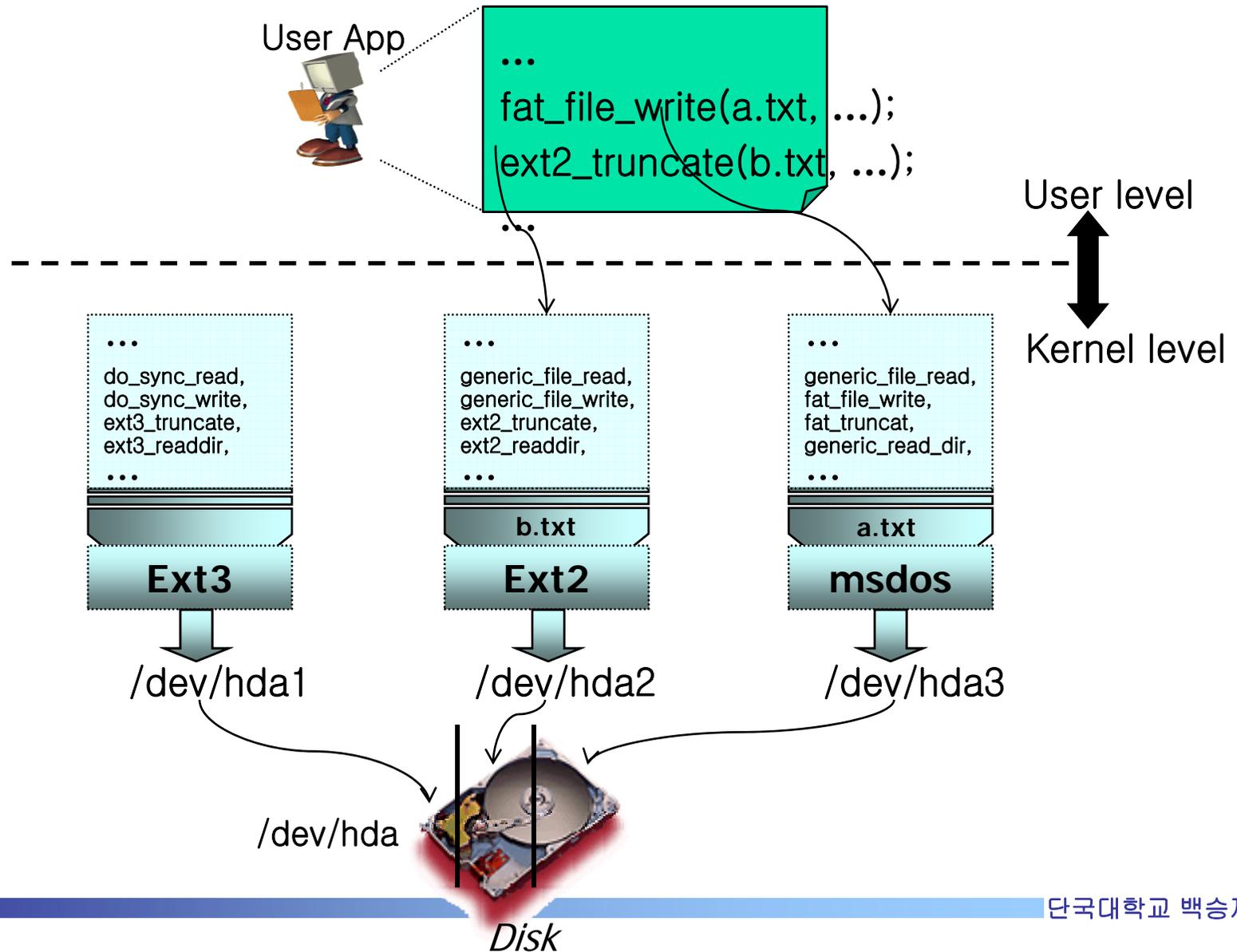
- create, read, write ... 의 함수는?
- cp ./a.txt /b.txt

```
Int cp( ... )
{
    int a, b, rb;
    char buf[SIZE_OF_BLOCK];
    ...
    a = MYFS_file_open(./a.txt);
    b = EXT2_file_open(/b.txt);
    rb = MYFS_file_read(a, buf);
    while( rb ){
        EXT2_file_write(b, rb, buf);
        ...
        rb = MYFS_file_read(a, buf);
        ...
    }
    ...
}
```

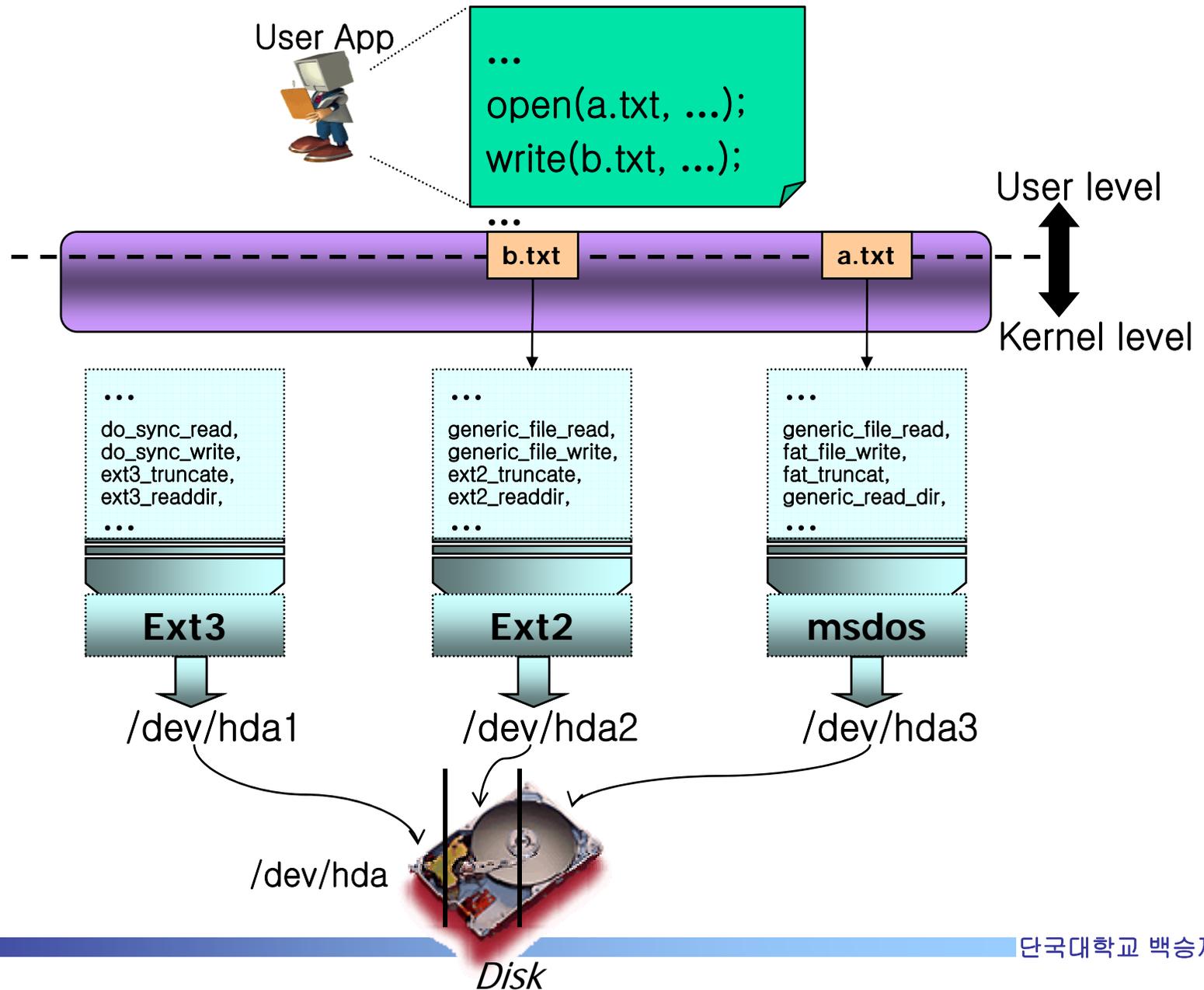
정말 cp 프로그램을 이렇게 만들어 줘야 할까?

이렇게 FS 종속적이지 않도록 하려면?

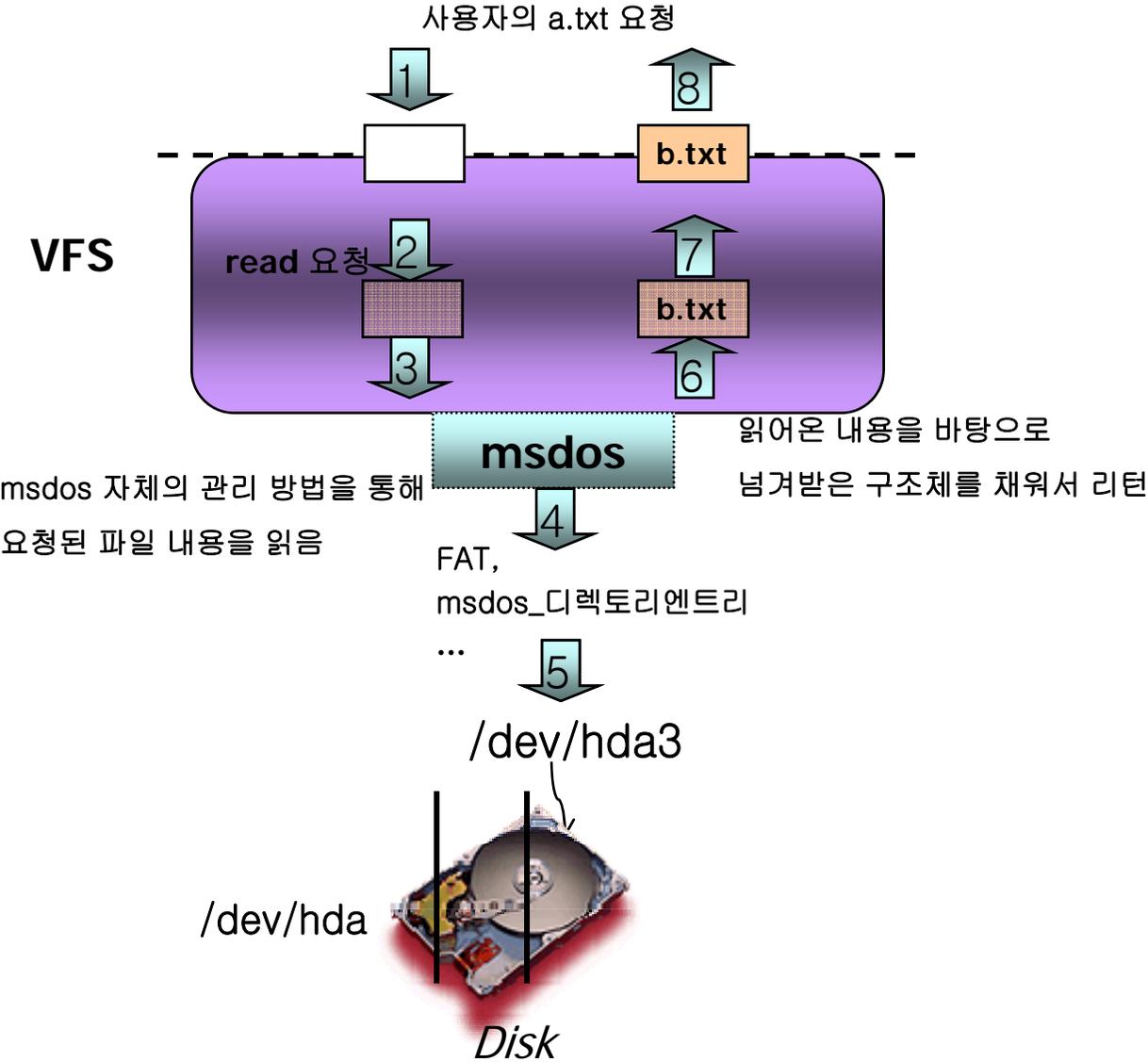
다양한 파일시스템 지원의 문제점



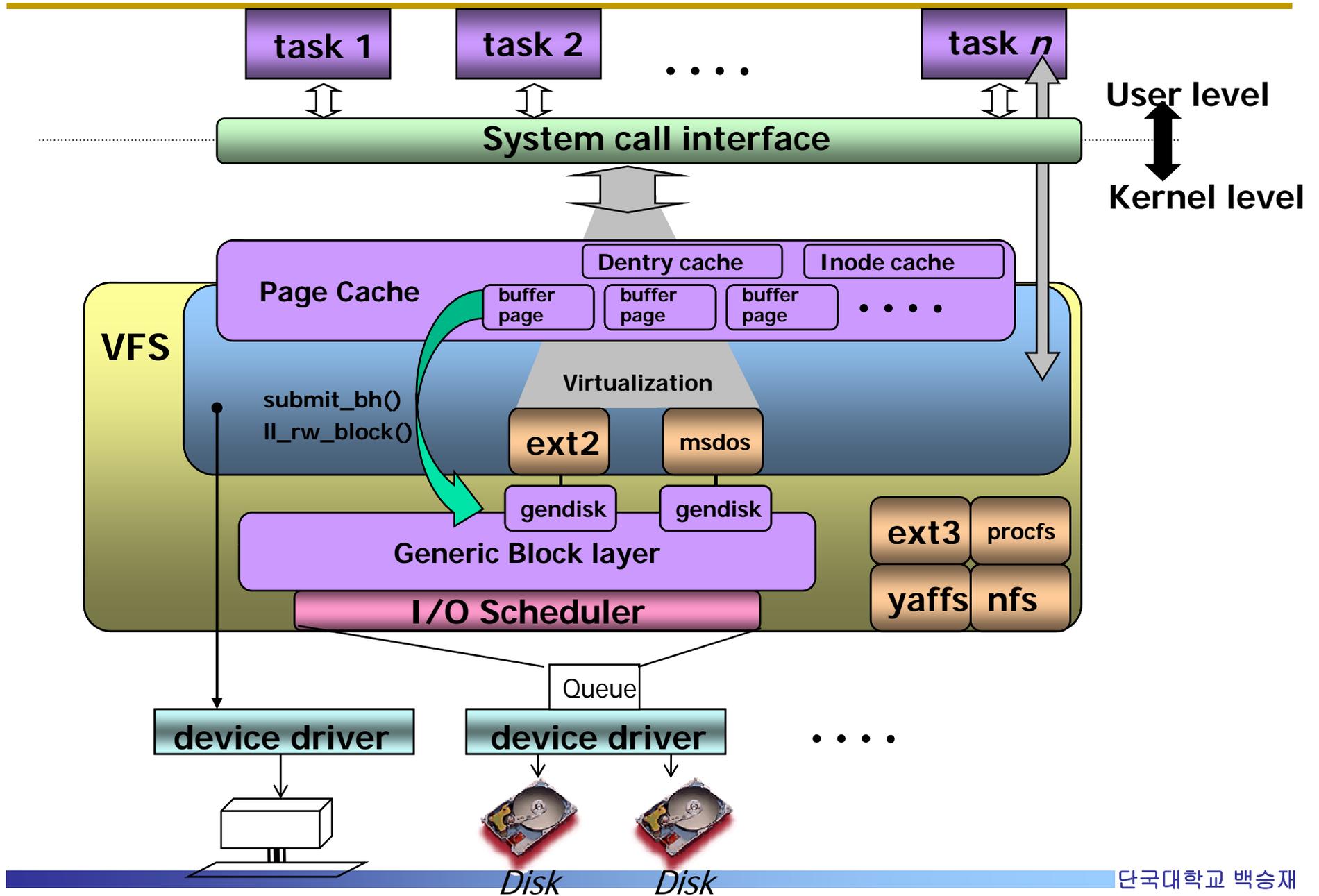
가상적인(Virtual) 계층의 도입



VFS의 동작 원리



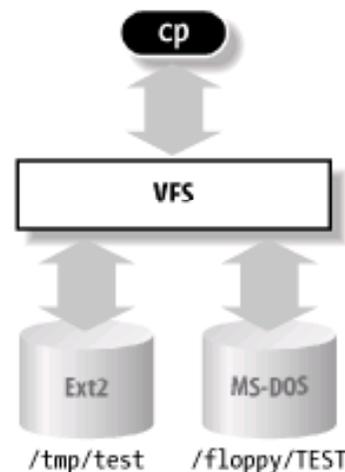
Linux 파일 시스템 구조



VFS

■ VFS(Virtual File System)의 역할

- ✓ 표준 UNIX FS이 제공하는 모든 시스템 콜을 처리하는 Kernel S/W 계층
- ✓ 여러 종류의 FS에 대해 공통 Interface를 제공
- ✓ 종류
 - 디스크 기반 FS
 - Ext2, Ext3, ReiserFS
 - SYSV, UFS, MINIX, VxFS
 - VFAT, NTFS
 - ISO 9660 CD-ROM FS, UDF
 - HPFS, HFS, AFFS, ADFS
 - JFS, XFS
 - 네트워크 FS
 - 특수 FS(가상 FS)



```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
           O_WRONLY|O_CREATE|O_TRUNC, 0600);
do {
    l = read(inf, buf, 4096);
    write(outf, buf, l);
} while (l);
close(outf);
close(inf);
```

■ VFS가 처리하는 system call

System call name	Description
<code>mount()</code> <code>umount()</code>	Mount/unmount filesystems
<code>sysfs()</code>	Get filesystem information
<code>statfs()</code> <code>fstatfs()</code> <code>ustat()</code>	Get filesystem statistics
<code>chroot()</code> <code>pivot_root()</code>	Change root directory
<code>chdir()</code> <code>fchdir()</code> <code>getcwd()</code>	Manipulate current directory
<code>mkdir()</code> <code>rmdir()</code>	Create and destroy directories
<code>getdents()</code> <code>readdir()</code> <code>link()</code> <code>unlink()</code> <code>rename()</code>	Manipulate directory entries
<code>readlink()</code> <code>symlink()</code>	Manipulate soft links
<code>chown()</code> <code>fchown()</code> <code>lchown()</code>	Modify file owner
<code>chmod()</code> <code>fchmod()</code> <code>utime()</code>	Modify file attributes
<code>stat()</code> <code>fstat()</code> <code>lstat()</code> <code>access()</code>	Read file status
<code>open()</code> <code>close()</code> <code>creat()</code> <code>umask()</code>	Open and close files
<code>dup()</code> <code>dup2()</code> <code>fcntl()</code>	Manipulate file descriptors
<code>select()</code> <code>poll()</code>	Asynchronous I/O notification
<code>truncate()</code> <code>ftruncate()</code>	Change file size
<code>lseek()</code> <code>_llseek()</code>	Change file pointer
<code>read()</code> <code>write()</code> <code>readv()</code> <code>writev()</code> <code>sendfile()</code> <code>readahead()</code>	Carry out file I/O operations
<code>pread()</code> <code>pwrite()</code>	Seek file and access it
<code>mmap()</code> <code>munmap()</code> <code>madvise()</code> <code>mincore()</code>	Handle file memory mapping
<code>fdatasync()</code> <code>fsync()</code> <code>sync()</code> <code>msync()</code>	Synchronize file data
<code>flock()</code>	Manipulate file lock

■ VFS의 객체 유형

✓ 슈퍼블록 객체

- 마운트 된 FS을 나타내는 정보 저장
- (디스크의) 파일 시스템 제어 블록(filesystem control block)에 대응

✓ 아이노드 객체

- 특정 파일을 나타내는 정보 저장
- (디스크의) 파일 제어 블록(file control block)에 대응
- inode 번호로 FS 내에서 유일하게 식별됨

✓ 파일 객체

- 프로세스와 연관되어 Open되어있는 파일과 관련된 정보를 저장
- 각 process 가 file에 접근하는 동안 커널 메모리에만 존재

✓ 디엔트리 객체

- 디렉토리 항목과 이에 대응하는 파일의 연결에 대한 정보를 저장
- 독자적 방법으로 디스크에 저장함

■ 디엔트리 캐시

- ✓ 가장 최근에 사용한 디엔트리 객체를 캐시

VFS의 Super Block 객체

■ VFS 자료 구조

✓ 슈퍼블록 객체(struct super_block, <linux/fs.>)

Type	Field	Description
struct list_head	s_list	Pointers for superblock list
kdev_t	s_dev	Device identifier
unsigned long	s_blocksize	Block size in bytes
unsigned char	s_blocksize_bits	Block size in number of bits
unsigned char	s_dirt	Modified (dirty) flag
unsigned long long	s_maxbytes	Maximum size of the files
struct file_system_type *	s_type	Filesystem type
struct super_operations *	s_op	Superblock methods
struct dqquot_operations *	dq_op	Disk quota methods
unsigned long	s_flags	Mount flags
unsigned long	s_magic	Filesystem magic number
struct dentry *	s_root	Dentry object of mount directory
struct rw_semaphore	s_umount	Semaphore used for unmounting
struct semaphore	s_lock	Superblock semaphore
int	s_count	Reference counter
atomic_t	s_active	Secondary reference counter
struct list_head	s_dirty	List of modified inodes
struct list_head	s_locked_inodes	List of inodes involved in I/O
struct list_head	s_files	List of file objects assigned to the superblock
struct block_device *	s_bdev	Pointer to the block device driver descriptor
struct list_head	s_instances	Pointers for a list of superblock objects of a given filesystem type
struct quota_mount_options	s_dquot	Options for disk quota
union	u	Specific filesystem information

Super Block 객체 관련 연산

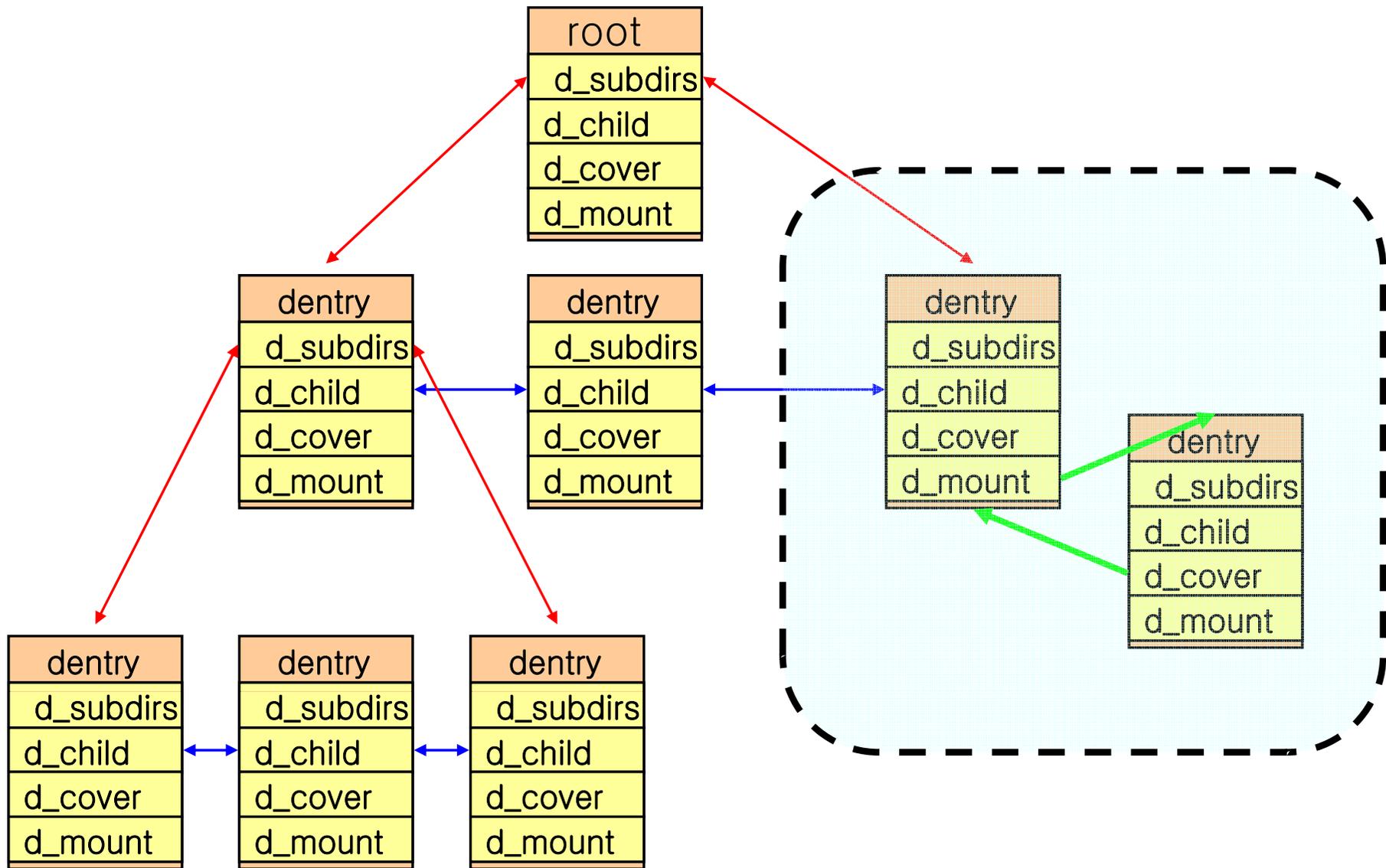
```
struct super_operations {
    struct inode *(*alloc_inode)(struct super_block *sb);
    void (*destroy_inode)(struct inode *);

    void (*read_inode) (struct inode *);

    void (*read_inode2) (struct inode *, void *) ;
    void (*dirty_inode) (struct inode *);
    void (*write_inode) (struct inode *, int);
    void (*put_inode) (struct inode *);
    void (*delete_inode) (struct inode *);
    void (*put_super) (struct super_block *);
    void (*write_super) (struct super_block *);
    int (*sync_fs) (struct super_block *);
    void (*write_super_lockfs) (struct super_block *);
    void (*unlockfs) (struct super_block *);
    int (*statfs) (struct super_block *, struct statfs *);
    int (*remount_fs) (struct super_block *, int *, char *);
    void (*clear_inode) (struct inode *);
    void (*umount_begin) (struct super_block *);

    struct dentry * (*fh_to_dentry)(struct super_block *sb, __u32 *fh, int len, int fhstype, int parent);
    int (*dentry_to_fh)(struct dentry *, __u32 *fh, int *lenp, int need_parent);
    int (*show_options)(struct seq_file *, struct vfsmount *);
};
```

VFS' mounted FS system structure



Inode 객체

✓ 아이노드 객체(struct inode <linux/fs.h>)

Type	Field	Description
struct list_head	i_hash	Pointers for the hash list
struct list_head	i_list	Pointers for the inode list
struct list_head	i_dentry	Pointers for the dentry list
struct list_head	i_dirty_buffers	Pointers for the modified buffers list
struct list_head	i_dirty_data_buffers	Pointers for the modified data buffers list
unsigned long	i_ino	inode number
unsigned int	i_count	Usage counter
kdev_t	i_dev	Device identifier
umode_t	i_mode	File type and access rights
nlink_t	i_nlink	Number of hard links
uid_t	i_uid	Owner identifier
gid_t	i_gid	Group identifier
kdev_t	i_rdev	Real device identifier
off_t	i_size	File length in bytes
time_t	i_atime	Time of last file access
time_t	i_mtime	Time of last file write
time_t	i_ctime	Time of last inode change
unsigned int	i_blkbits	Block size in number of bits
unsigned long	i_blksize	Block size in bytes
unsigned long	i_blocks	Number of blocks of the file
unsigned long	i_version	Version number, automatically incremented after each use
struct semaphore	i_sem	inode semaphore
struct semaphore	i_zombie	Secondary inode semaphore used when removing or renaming the inode

<code>struct inode_operations *</code>	<code>i_op</code>	inode operations
<code>struct file_operations *</code>	<code>i_fop</code>	Default file operations
<code>struct super_block *</code>	<code>i_sb</code>	Pointer to superblock object
<code>wait_queue_head_t</code>	<code>i_wait</code>	inode wait queue
<code>struct file_lock *</code>	<code>i_flock</code>	Pointer to file lock list
<code>struct address_space *</code>	<code>i_mapping</code>	Pointer to an <code>address_space</code> object (see Chapter 14)
<code>struct address_space</code>	<code>i_data</code>	<code>address_space</code> object for block device file
<code>struct dquot **</code>	<code>i_dquot</code>	inode disk quotas
<code>struct list_head</code>	<code>i_devices</code>	Pointers of a list of block device file inodes (see Chapter 13)
<code>struct pipe_inode_info *</code>	<code>i_pipe</code>	Used if the file is a pipe (see Chapter 19)
<code>struct block_device *</code>	<code>i_bdev</code>	Pointer to the block device driver
<code>struct char_device *</code>	<code>i_cdev</code>	Pointer to the character device driver
<code>unsigned long</code>	<code>i_dnotify_mask</code>	Bit mask of directory notify events
<code>struct dnotify_struct *</code>	<code>i_dnotify</code>	Used for directory notifications
<code>unsigned long</code>	<code>i_state</code>	inode state flags
<code>unsigned int</code>	<code>i_flags</code>	Filesystem mount flags
<code>unsigned char</code>	<code>i_sock</code>	Nonzero if file is a socket
<code>atomic_t</code>	<code>i_writecount</code>	Usage counter for writing processes
<code>unsigned int</code>	<code>i_attr_flags</code>	File creation flags
<code>__u32</code>	<code>i_generation</code>	inode version number (used by some filesystems)
<code>union</code>	<code>u</code>	Specific filesystem information

Inode 객체 관련 연산

- inode 관련 연산은 inode_operations 구조체에 따라 i_op 필드에

```

struct inode_operations {
    int (*create) (struct inode *, struct dentry *, int);
    struct dentry * (*lookup) (struct inode *, struct dentry *);
    int (*link) (struct dentry *, struct inode *, struct dentry *);
    int (*unlink) (struct inode *, struct dentry *);
    int (*symlink) (struct inode *, struct dentry *, const char *);
    int (*mkdir) (struct inode *, struct dentry *, int);
    int (*rmdir) (struct inode *, struct dentry *);
    int (*mknod) (struct inode *, struct dentry *, int, int);
    int (*rename) (struct inode *, struct dentry *,
                  struct inode *, struct dentry *);
    int (*readlink) (struct dentry *, char *, int);
    int (*follow_link) (struct dentry *, struct nameidata *);
    void (*truncate) (struct inode *);
    int (*permission) (struct inode *, int);
    int (*revalidate) (struct dentry *);
    int (*setattr) (struct dentry *, struct iattr *);
    int (*getattr) (struct dentry *, struct iattr *);
    int (*setxattr) (struct dentry *, const char *, void *, size_t, int);
    ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
    ssize_t (*listxattr) (struct dentry *, char *, size_t);
    int (*removexattr) (struct dentry *, const char *);
};

```

✓ 파일 객체(struct file <linux/fs.h>)

Type	Field	Description
struct list_head	f_list	Pointers for generic file object list
struct dentry *	f_dentry	dentry object associated with the file
struct vfsmount *	f_vfsmnt	Mounted filesystem containing the file
struct file_operations *	f_op	Pointer to file operation table
atomic_t	f_count	File object's usage counter
unsigned int	f_flags	Flags specified when opening the file
mode_t	f_mode	Process access mode
loff_t	f_pos	Current file offset (file pointer)
unsigned long	f_reada	Read-ahead flag
unsigned long	f_ramax	Maximum number of pages to be read-ahead
unsigned long	f_raend	File pointer after last read-ahead
unsigned long	f_ralen	Number of read-ahead bytes
unsigned long	f_rawin	Number of read-ahead pages
struct fown_struct	f_owner	Data for asynchronous I/O via signals
unsigned int	f_uid	User's UID
unsigned int	f_gid	User's GID
int	f_error	Error code for network write operation
unsigned long	f_version	Version number, automatically incremented after each use
void *	private_data	Needed for tty driver
struct kiobuf *	f_iobuf	Descriptor for direct access buffer (see Section 15.2)
long	f_iobuf_lock	Lock for direct I/O transfer

File 객체 관련 연산

- 파일에 대한 연산을 file_operations 구조체에 담아 f_op에

```

struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
};

```

Dentry 객체

- ✓ 디엔트리 객체(struct dentry <linux/dcache.h>)
 - 디렉토리 엔트리를 메모리로 읽은 뒤, VFS가 dentry 구조체의 디엔트리 객체로 변환
 - P가 탐색하는 경로명의 모든 구성 요소에 대해 디엔트리 객체 생성

Type	Field	Description
atomic_t	d_count	Dentry object usage counter
unsigned int	d_flags	Dentry flags
struct inode *	d_inode	Inode associated with filename
struct dentry *	d_parent	Dentry object of parent directory
struct list_head	d_hash	Pointers for list in hash table entry
struct list_head	d_lru	Pointers for unused list
struct list_head	d_child	Pointers for the list of dentry objects included in parent directory
struct list_head	d_subdirs	For directories, list of dentry objects of subdirectories
struct list_head	d_alias	List of associated inodes (alias)
int	d_mounted	Flag set to 1 if and only if the dentry is the mount point for a filesystem
struct qstr	d_name	Filename
unsigned long	d_time	Used by d_revalidate method
struct dentry_operations*	d_op	Dentry methods
struct super_block *	d_sb	Superblock object of the file
unsigned long	d_vfs_flags	Dentry cache flags
void *	d_fsdata	Filesystem-dependent data
unsigned char *	d_iname	Space for short filename

Dentry 객체 관련 연산

- dentry 객체에 대한 연산을 dentry_operations 구조체에 담아 d_op 필드에, 대개 NULL임

```

struct dentry {
    atomic_t d_count;
    unsigned int d_flags;
    struct inode * d_inode;           /* Where the name belongs to - NULL is negative */
    struct dentry * d_parent;        /* parent directory */
    struct list_head d_hash;         /* lookup hash list */
    struct list_head d_lru;          /* d_count = 0 LRU list */
    struct list_head d_child;        /* child of parent list */
    struct list_head d_subdirs;      /* our children */
    struct list_head d_alias;        /* inode alias list */
    int d_mounted;
    struct qstr d_name;
    unsigned long d_time;            /* used by d_revalidate */
    struct dentry_operations *d_op;
    struct super_block * d_sb;        /* The root of the dentry tree */
    unsigned long d_vfs_flags;
    void * d_fsdata;                 /* fs-specific data */
    unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
};

```

Task와 파일

■ 프로세스 관련 파일

- ✓ 각 process 디스크립터에 process의 fs_struct 구조체를 가리키는 fs 필드 유지
- ✓ struct fs_struct, <linux/fs_struct.h>
- ✓ 프로세스와 관련된 파일시스템 정보를 유지
 - 현재 작업 디렉토리와 현 프로세스의 루트 디렉토리 정보

Type	Field	Description
atomic_t	count	Number of processes sharing this table
rwlock_t	lock	Read/write spin lock for the table fields
int	umask	Bit mask used when opening the file to set the file permissions
struct dentry *	root	Dentry of the root directory
struct dentry *	pwd	Dentry of the current working directory
struct dentry *	altroot	Dentry of the emulated root directory (always NULL for the 80 x 86 architecture)
struct vfsmount *	rootmnt	Mounted filesystem object of the root directory
struct vfsmount *	pwdmnt	Mounted filesystem object of the current working directory
struct vfsmount *	altrootmnt	Mounted filesystem object of the emulated root directory (always NULL for the 80 x 86 architecture)

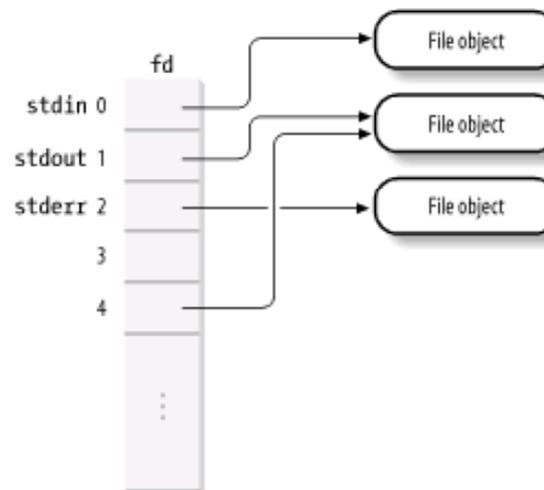
Task와 파일

- ✓ 각 process 디스크립터에, 각 process가 open한 file을 나타내는 files_struct 구조체를 가리키는 files 필드
- ✓ struct files_struct <linux/file.h>

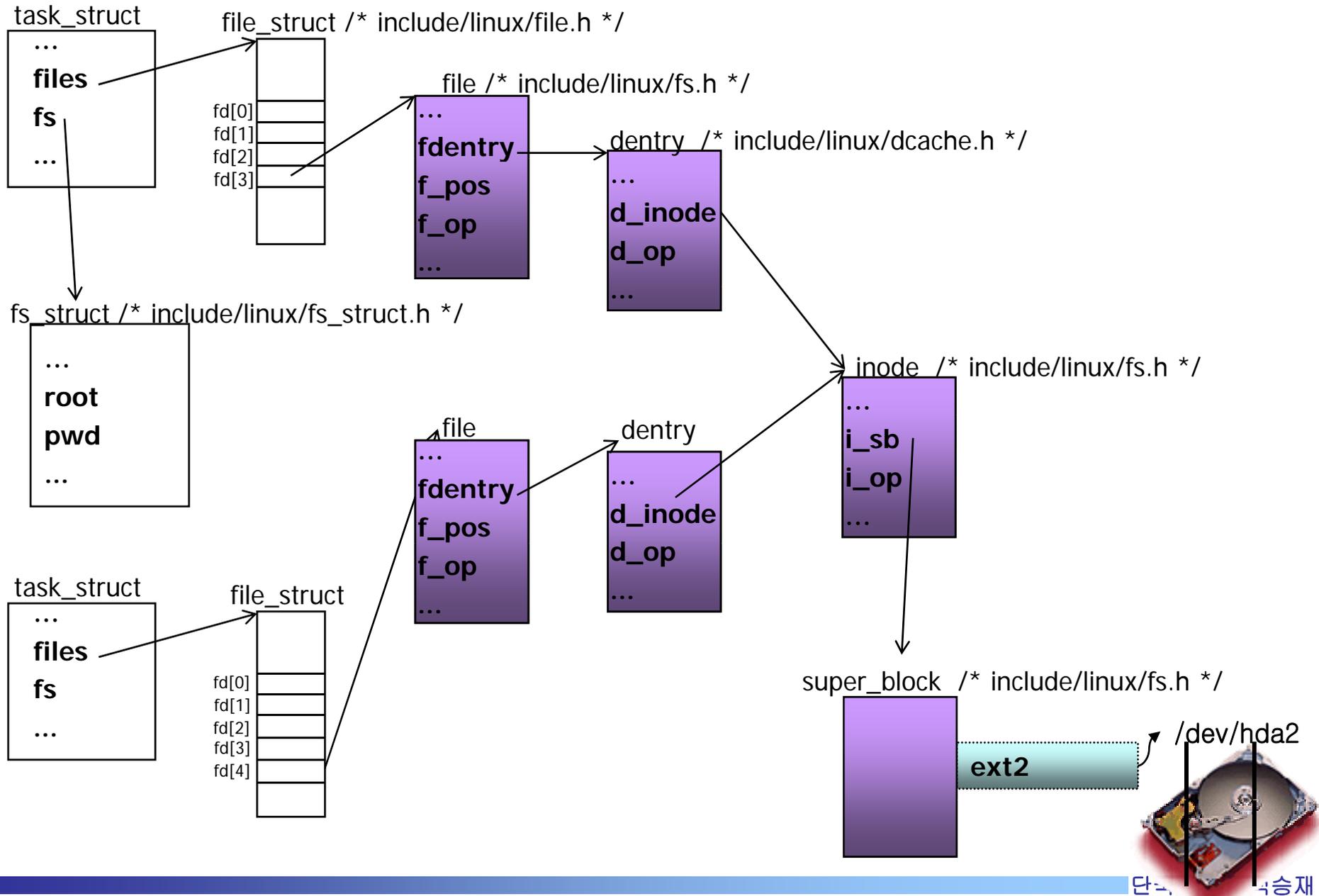
Type	Field	Description
atomic_t	count	Number of processes sharing this table
rwlock_t	file_lock	Read/write spin lock for the table fields
int	max_fds	Current maximum number of file objects
int	max_fdset	Current maximum number of file descriptors
int	next_fd	Maximum file descriptors ever allocated plus 1
struct file **	fd	Pointer to array of file object pointers
fd_set *	close_on_exec	Pointer to file descriptors to be closed on <code>exec()</code>
fd_set *	open_fds	Pointer to open file descriptors
fd_set	close_on_exec_init	Initial set of file descriptors to be closed on <code>exec()</code>
fd_set	open_fds_init	Initial set of file descriptors
struct file **	fd_array	Initial array of file object pointers

Task와 파일

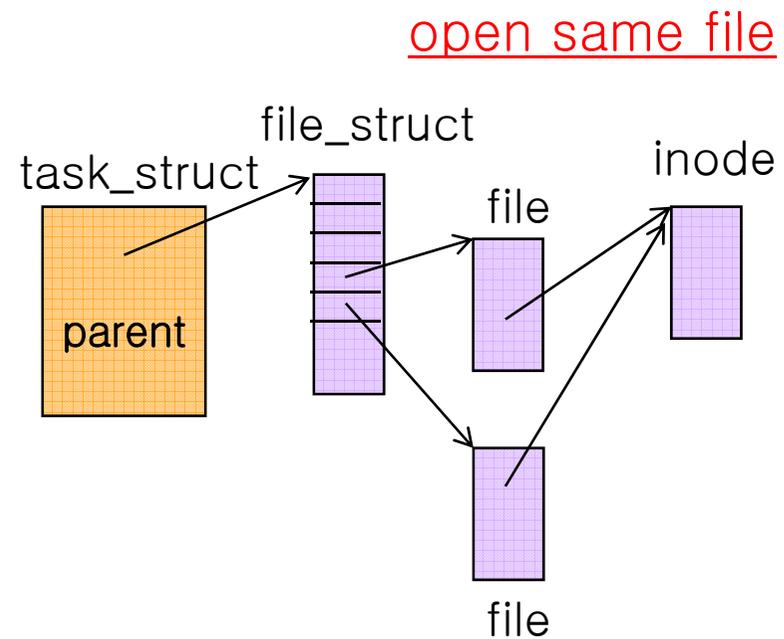
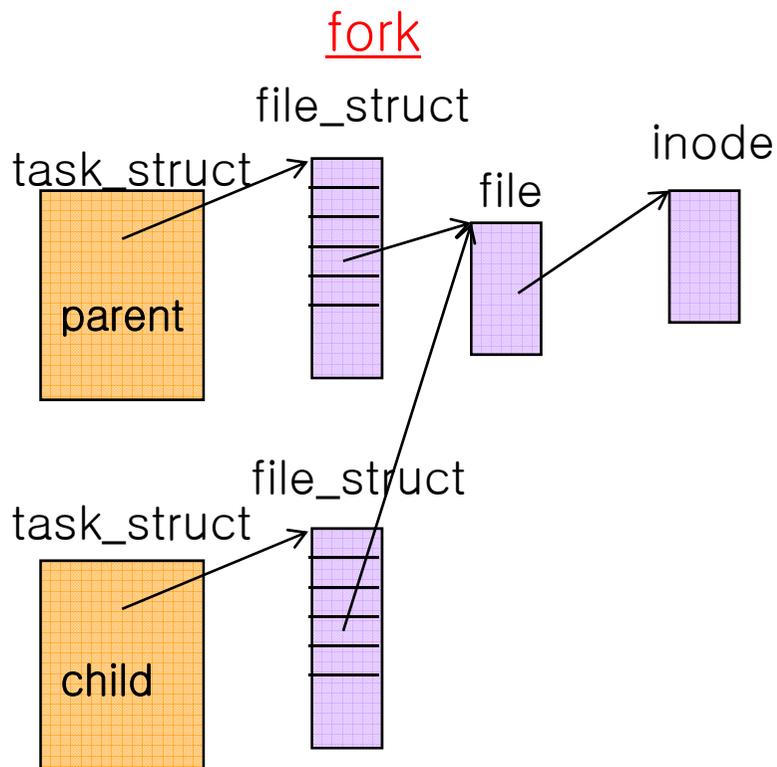
- fd : 파일 객체 포인터 배열(크기 = max_fds), files_struct 구조체의 fd_array(파일 객체 포인터 32개 담고 있음) 필드를 가리킴 → P가 32개 이상 열면, 더 큰 파일 ptr array를 할당, max_fds값 갱신
- fd_array 배열은 각 인덱스가 file descriptor
 - file descriptor
 - dup(), dup2(), fcntl()로 두 file descriptor가 같은 파일 가리키게 가능
 - 한 P는 NR_OPEN(1048576)개 이상 사용불가
 - P 디스크립터의 rlim[RLIMIT_NOFILE]구조체에 file descriptor의 최대 수에 대한 동적 제한(1024)가지고 있음
- open_fds : open_fds_init 필드의 주소 가짐 → 현재 열린 파일의 file descriptor를 식별하는 비트맵(max_fdset개)
- fd_set : 1024비트 자료구조, 확장 필요 시 동적으로 확장함



task_struct와 VFS의 객체



■ 커널 자료 구조 관계



☒ how about **dup**?

Example - 간단한 FS실습(1/2)

57

```
[root@embeddedDell root]# dd if=/dev/zero of=./img bs=1k count=8192
8192+0개의 레코드를 입력하였습니다
8192+0개의 레코드를 출력하였습니다
[root@embeddedDell root]# mke2fs ./img
mke2fs 1.32 (09-Nov-2002)
./img is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2048 inodes, 8192 blocks
409 blocks (4.99%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
2048 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

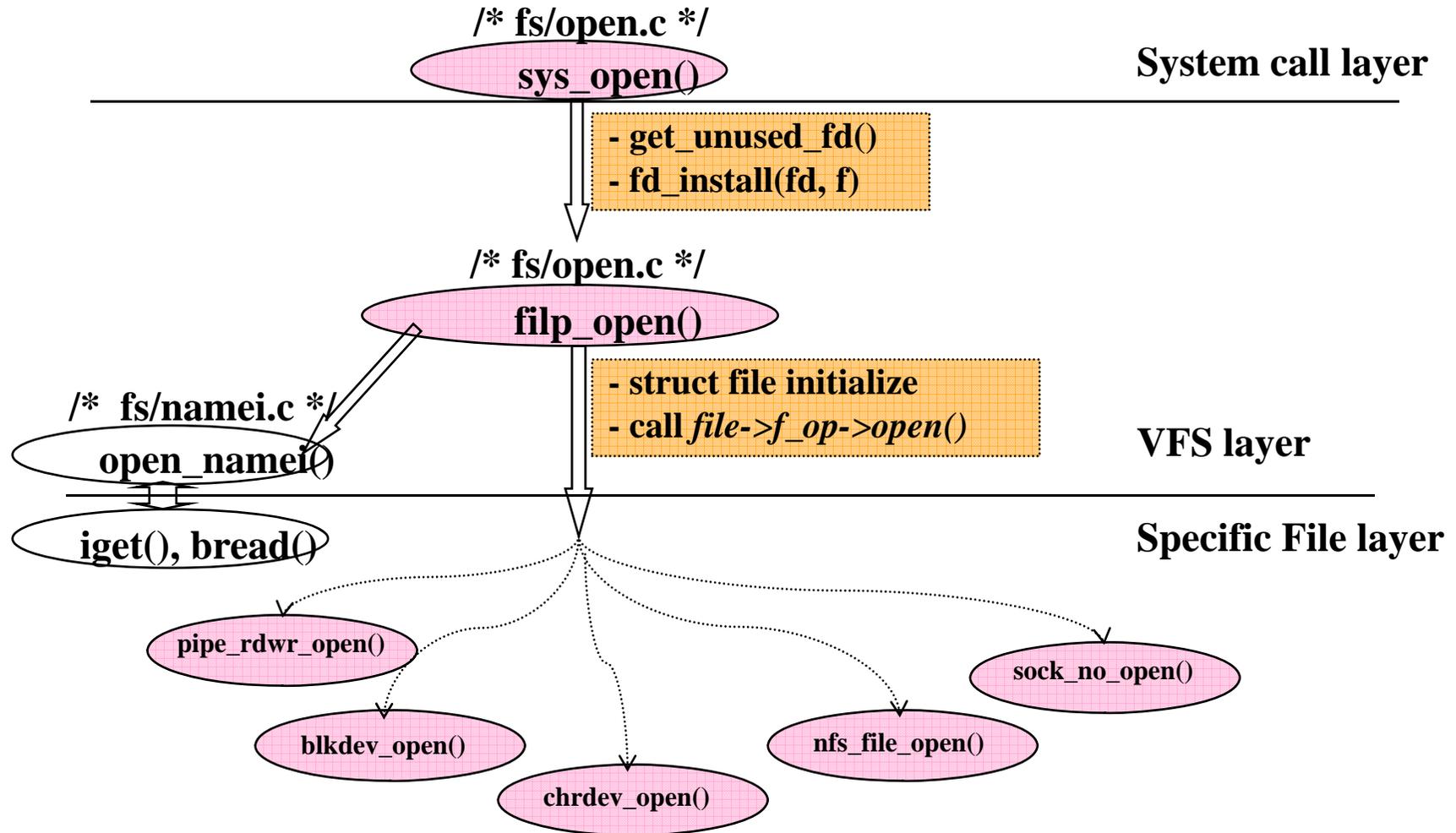
This filesystem will be automatically checked every 21 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
[root@embeddedDell root]#
```

Example - 간단한 FS실습(2/2)

```
[root@embeddedDell root]# mkdir mnt
[root@embeddedDell root]# mount -t ext2 -o loop ./img mnt
[root@embeddedDell root]#
[root@embeddedDell root]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda2       38G   7.5G   29G   21% /
/dev/hda1       99M   13M    82M   13% /boot
/dev/hdb2       73G   65G   4.4G   94% /home
none            251M   0     251M   0% /dev/shm
/root/img       7.8M   13K   7.4M   1% /root/mnt
[root@embeddedDell root]# cd mnt
[root@embeddedDell mnt]# ls -alh
합계 17K
drwxr-xr-x   3 root   root   1.0K   5월 30 12:06 .
drwxr-x---  51 root   root   4.0K   5월 30 12:07 ..
drwx-----  2 root   root   12K   5월 30 12:06 lost+found
[root@embeddedDell mnt]# touch a
[root@embeddedDell mnt]# ls
a  lost+found
[root@embeddedDell mnt]#
```

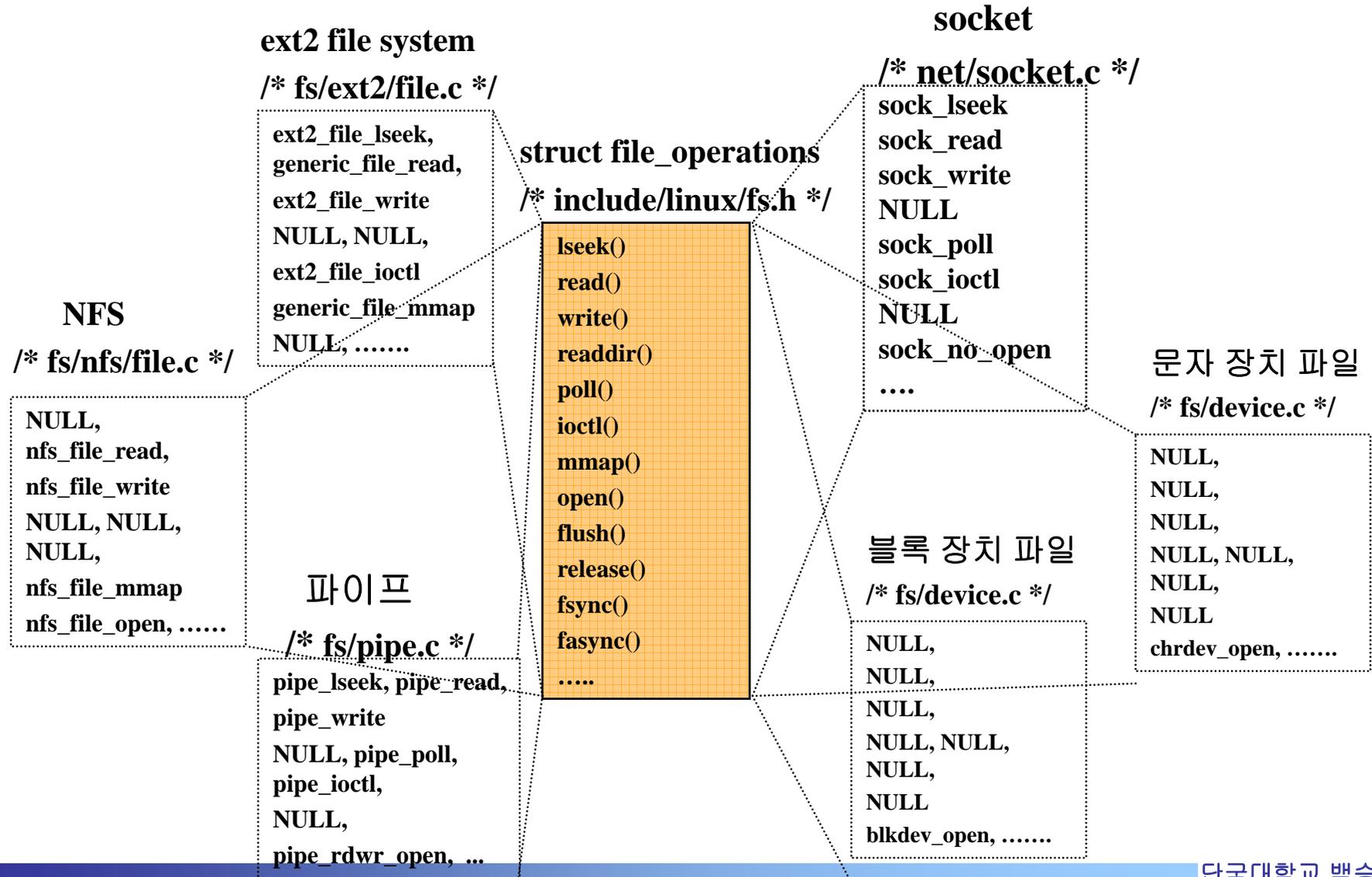
- Block Size를 2048 혹은, 4096으로 만들 수 있는가?

■ open 시스템 호출 분석



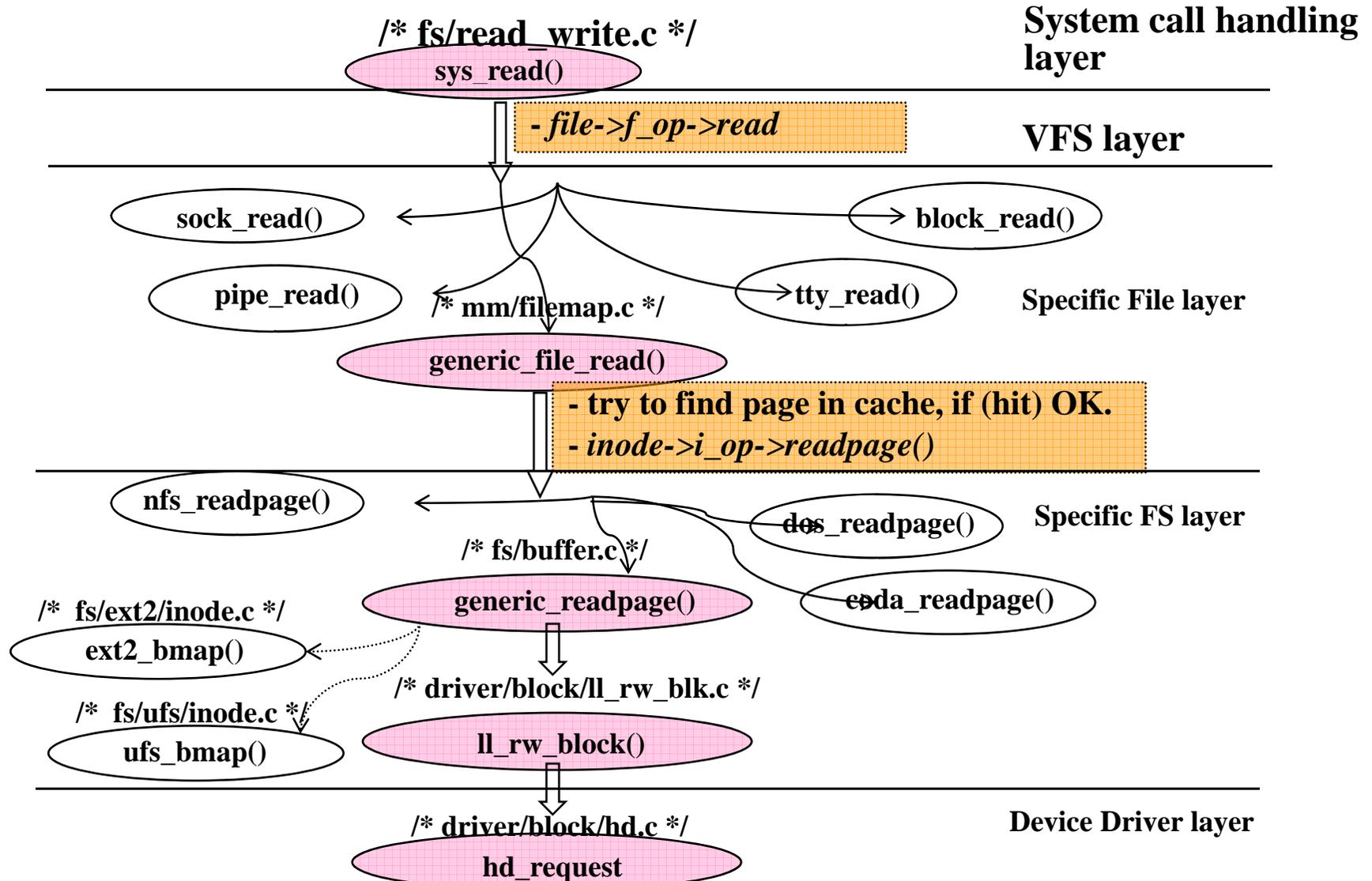
Linux의 파일 연산 구현 분석 (2/4)

- struct file_operations 구조 : 다양한 파일 유형을 일관된 인터페이스로 지원



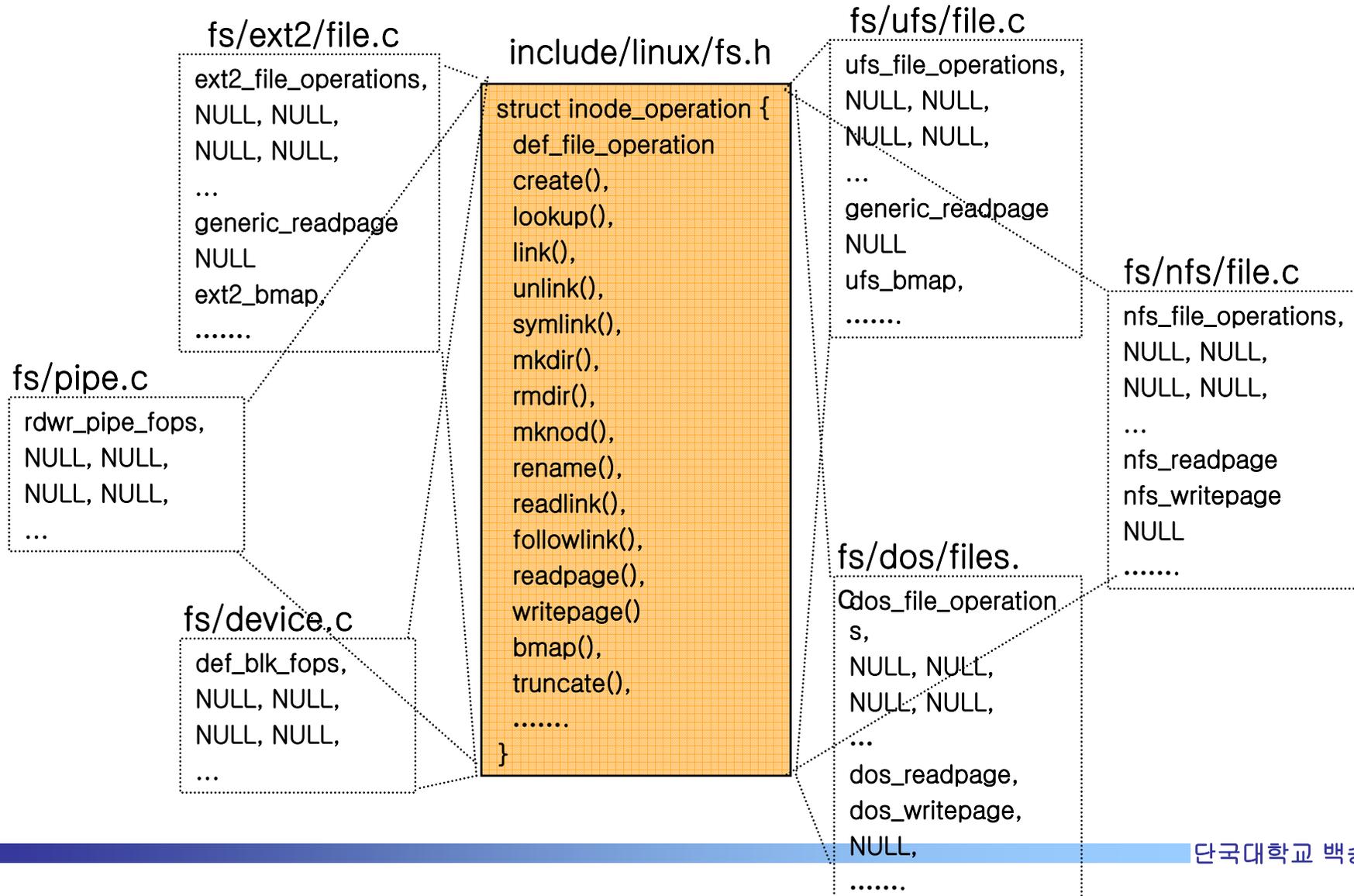
Linux의 파일 연산 구현 분석 (3/4)

■ open 시스템 호출 분석



Linux의 파일 연산 구현 분석 (4/4)

- struct inode_operations 구조 : 다양한 파일 시스템을 일관된 인터페이스로 지원



Example(1/3)– Lock 실험

- Process A에서 `fcntl()`이나 `flock()`함수를 사용하여 파일 `a.txt`에 lock을 건다
- Process A가 `fork()`하여 Process A'를 생성한다
- Process A'가 파일 `a.txt`에 접근 하려 하면?

- Process A가 파일 `a.txt`에 lock을 잡고 있는 동안, 창을 하나 더 띄워서 `vi` 편집기로 `a.txt`를 편집하려고 하면?

Example(2/3)– Lock 실험

- Advisory lock(POSIX : `fcntl()`, BSD : `flock()`)
 - OS가 특정 프로세스에 의해 Lock된 파일에 대하여 정확한 자료를 유지하되, 다른 프로세스에 의해 Lock된 파일에 대하여 쓰기를 막지 않는 것을 말함
 - 프로세스는 advisory lock을 무시하고 적절한 권한이 있다면 Lock된 파일에 쓰기를 할 수 있음
- Mandatory lock(System V Release 3 : `lockf()`)
 - 호출 프로세스가 접근하려는 파일에 대해 lock을 위반하지 않는가를 검사하기 위해 kernel로 하여금 open, read, write를 검사하는 방식

Example(3/3)– Lock 실험

■ File locking on linux

✓ Linux에서 locking

- Advisory locks
 - `fcntl()` – `FL_POSIX` 타입
 - `flock()` – `FL_LOCK` 타입
- mandatory locks
 - `lockf()` – System V
- `mount()` 시스템 콜의 `MS_MANDLOCK` flag를 사용하여 파일시스템 당 enable와 disable할 수 있음
- 디폴트 값은 mandatory locking switch off
- 두 가지 lock은 공존하며 서로에게 영향을 주지 않음
- `fcntl()`이 file에 lock을 걸면 `flock()`는 나타나지 않음