CSE 306 Operating Systems
Linux Virtual Filesystem

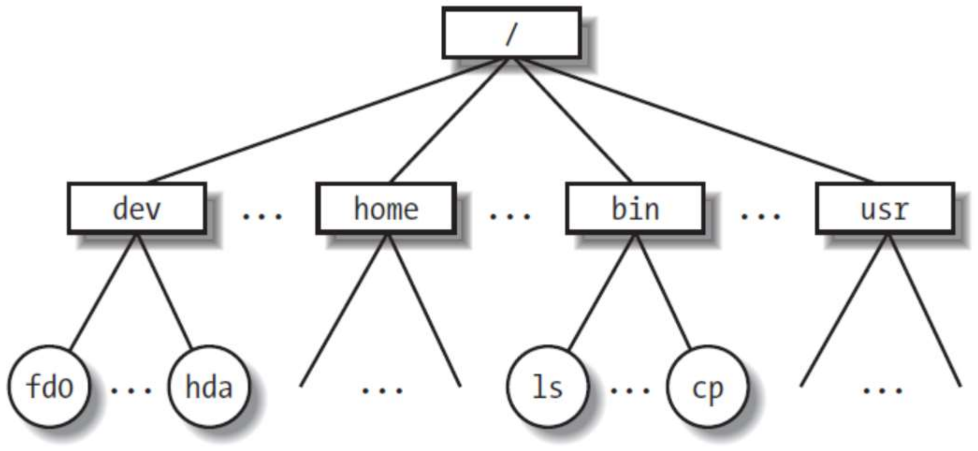
YoungMin Kwon

Filesystem

- Filesystem
  - A hierarchical storage of data adhering to a specific structure
  - Filesystems control how data is stored and retrieved
  - Filesystems contain files, directories, and associated control information
  - Typical filesystem operations
    - creation, deletion, mounting, …

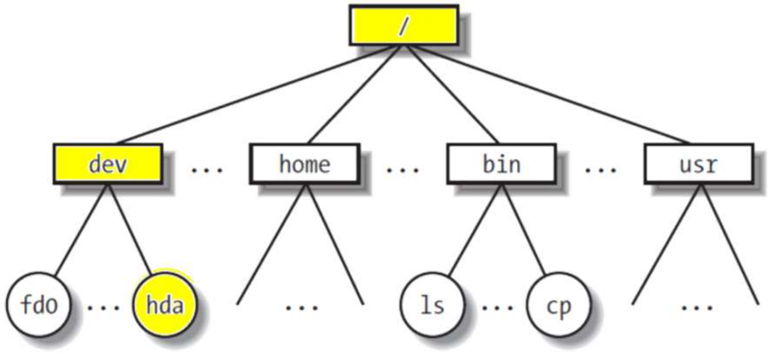
Filesystem

- File
  - A file is an ordered string of bytes.
  - Each file is assigned a human-readable name
  - Typical file operations
    - Read, write, create, delete, …
- Directory
  - A directory is a file that lists the files contained therein
  - A directory can contain subdirectories



Filesystem

- Path
  - Directories may be nested to form a path
- Directory entry (dentry)
  - Each component of a path
  - Has pointers to
    - name, inode, parent dentry



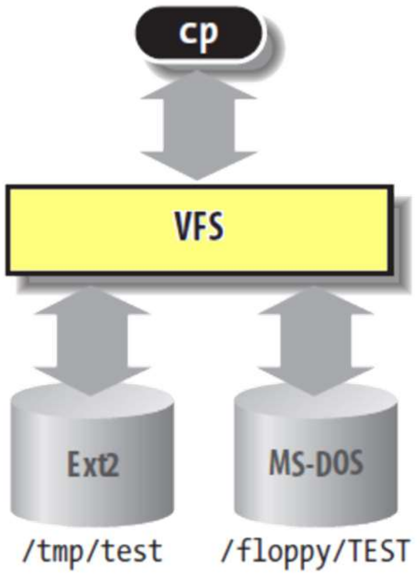
- Example
  - /dev/hda is a path
  - The rootdiectory /, directories dev, and the file hda are all dentries

Filesystem

- File metadata
  - Any associated information about a file
    - access permission, size, owner, creation time, …
  - File metadata is stored in a separate data structure from a file, called the inode (index node)
- Superblock
  - Metadata for filesystem: a data structure containing information about the filesystem as a whole

Common File System Interface

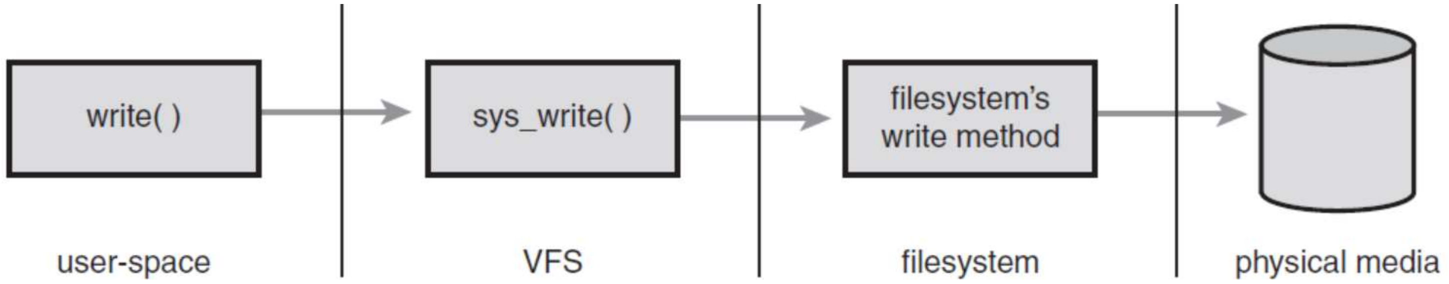
- Virtual File System (VFS)
  - A kernel software layer that handles all system calls related to a standard Unix file system
    - open(), read(), write(), …
  - Provides a common interface to several kinds of file systems



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

Common File System Interface

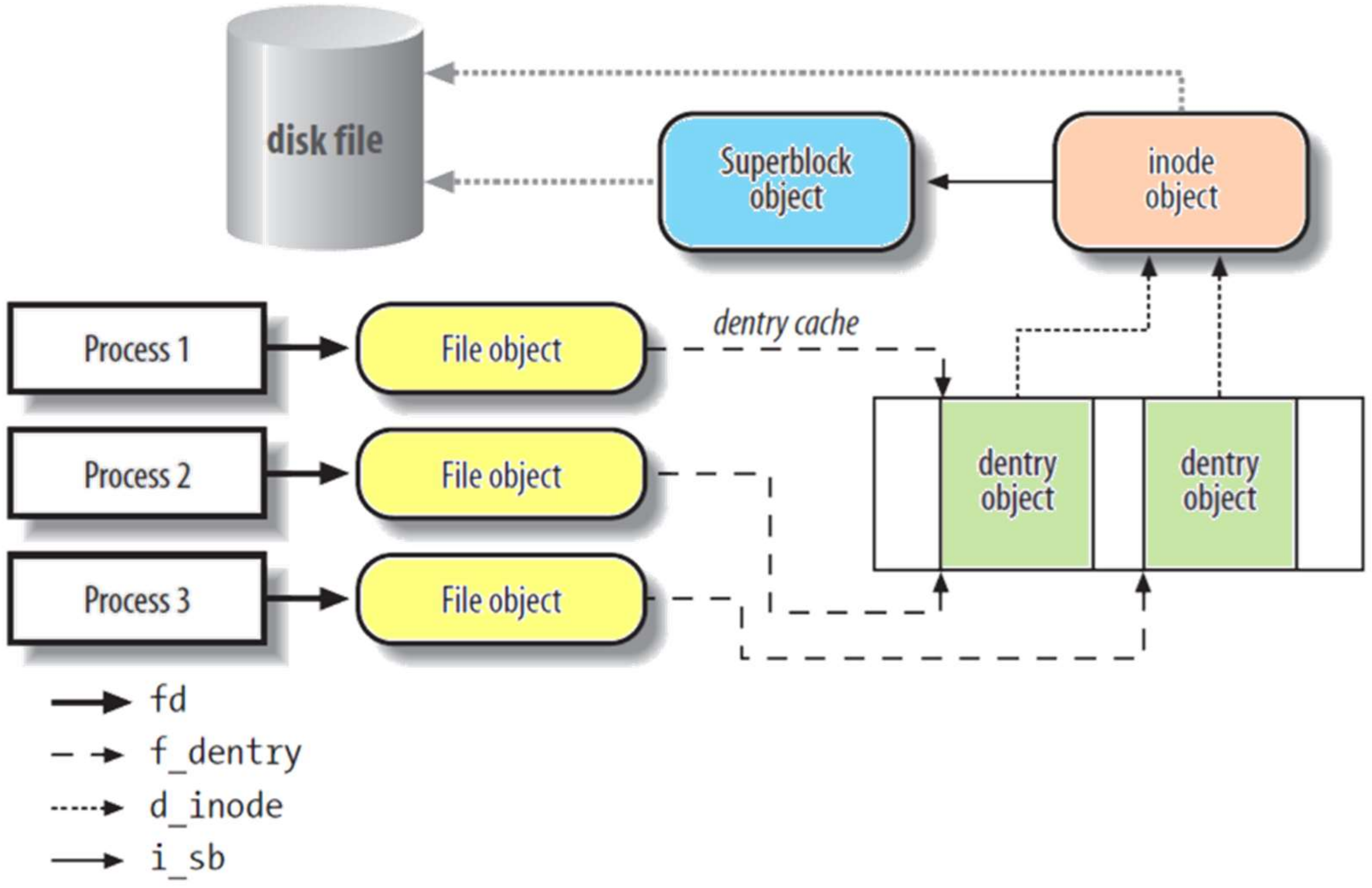
- Filesystem abstraction layer
  - VFS provides an abstraction layer by defining a conceptual interface and data structures that all filesystem support
  - `ret = write(fd, buf, len);`



VFS Objects

- Four primary object types
  - superblock object: represents a specific mounted filesystem
  - inode object: represents a specific file
  - dentry object: represents a single component of a path
    - About the linking of a directory entry with the corresponding file
  - file object: represents an open file associated with a process
    - About the interaction between an open file and a process

VFS Objects



- Interaction between processes and VFS objects

Superblock Object

- struct super_block
  - Implemented by each filesystem to store information describing the file system

- struct super_operations
  - Each item in this structure is a pointer to a function that operates on a superblock object
  - e.g. `sb->s_op->write_super(sb);`

Some of super_block Fields

```c
struct super_block {
    struct list_head s_list;            /* list of all superblocks */
    dev_t s_dev;                        /* identifier */
    unsigned long s_blocksize;          /* block size in bytes */
    unsigned char s_dirt;               /* dirty flag */
    struct file_system_type *s_type;  /* filesystem type */
    struct super_operations *s_op;    /* superblock methods */
    unsigned long s_flags;              /* mount flags */
    unsigned long s_magic;              /* filesystem’s magic number */
    struct dentry *s_root;              /* directory mount point */
    int s_count;                        /* superblock ref count */
    int s_need_sync;                    /* not-yet-synced flag */
    struct list_head s_inodes;          /* list of inodes */
    struct list_head s_dirty;           /* list of dirty inodes */
    fmode_t s_mode;                     /* mount permissions */
...
};
```

Some of super_operations Fields

```
struct super_operations {
    //create and initialize a new inode
    struct inode *(*alloc_inode)(struct super_block *sb);

    //deallocate the inode
    void(*destroy_inode)(struct inode *inode);

    // called when the inode is dirtied
    void(*dirty_inode) (struct inode *inode);

    // write the inode to disk
    int(*write_inode) (struct inode *inode, int wait);

    // delete the inode from the disk
    void(*delete_inode) (struct inode *inode);

    // called on unmount to release the superblock
    void(*put_super) (struct super_block *sb);

    // update the on-disk superblock with sb
    void(*write_super) (struct super_block *sb);
...
};
```

myfs: Register a Filesystem

```
// myfs.c
#include <linux/tty.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/slab.h>

#define IF_TRUE_GOTO(expr, label) {                          \
    if((expr)) {                                             \
        printk("error: %s, %s, %d\n",                        \
            __FILE__, __FUNCTION__, __LINE__);               \
        goto label;                                          \
    }                                                        \
}
#define IF_FALSE_GOTO(expr, label)                           \
    IF_TRUE_GOTO(!(expr), label)
#define IF_NULL_GOTO(expr, label)                            \
    IF_TRUE_GOTO((expr)==NULL, label)
#define PENTER printk(KERN_INFO "entering %s\n", __FUNCTION__)

#define MYFS_MAGIC 0xBADBEEF
#define MAX_FILE_SIZE 4096
#define DEFAULT_MODE_FILE   (S_IFREG | 0666) //for inode->i_mode
#define DEFAULT_MODE_DIR    (S_IFDIR | 0555) //for inode->i_mode
```

myfs: Register a Filesystem

```
static struct file_system_type myfs_type = {
    .owner = THIS_MODULE,
    .kill_sb = kill_litter_super,
    //TODO: update name and mount fields
    //name: name of this filesystem e.g. "myfs"
    //mount: the function that mounts this filesystem
};

static int __init myfs_init(void) {
    PENTER;
    return register_filesystem(&myfs_type);
}
static void __exit myfs_exit(void) {
    PENTER;
    unregister_filesystem(&myfs_type);
}

MODULE_AUTHOR("YoungMin Kwon"); //Put your name here
MODULE_LICENSE("GPL");
module_init(myfs_init);
module_exit(myfs_exit);
```

myfs: Mount a Filesystem

```
static const struct super_operations myfs_super_ops = {
    .statfs = simple_statfs,
    //TODO: initialize destroy_inode field
};

static int myfs_fill_super(struct super_block *sb, void *data, int silent) {
    static struct tree_descr files[] = {{""}};
    PENTER;
    IF_TRUE_GOTO(simple_fill_super(sb, MYFS_MAGIC, files), fail);
    //TODO: update sb’s super_operation element

    IF_TRUE_GOTO(myfs_create_tree(sb), fail);
    return 0;
fail:
    return -1;
}

static struct dentry *myfs_mount(struct file_system_type *fs_type,
    int flags, const char *dev_name, void *data) {
    PENTER;
    return mount_single(fs_type, flags, data, myfs_fill_super);
}
```

file name

file contents

```c
static int myfs_create_tree(struct super_block *sb) {
    struct dentry *dir = sb->s_root;
    PENTER;
    IF_NULL_GOTO(myfs_create_file(sb, dir, "a", "Hello from /a\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "b", "Hello from /b\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "c", "Hello from /c\n"), fail);

    IF_NULL_GOTO(dir = myfs_create_dir(sb,  dir, "D"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "a", "Hello from /D/a\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "b", "Hello from /D/b\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "c", "Hello from /D/c\n"), fail);

    IF_NULL_GOTO(dir = myfs_create_dir(sb,  dir, "E"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "a", "Hello from /D/E/a\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "b", "Hello from /D/E/b\n"), fail);
    IF_NULL_GOTO(myfs_create_file(sb, dir, "c", "Hello from /D/E/c\n"), fail);
    return 0;
fail:
    return -1;
}

static void myfs_destroy_inode(struct inode *inode) {
    PENTER;
    //TODO: for a regular file (check inode->i_mode), free its i_private field
}
```

Inode Object

- struct inode
  - Represents all the information needed by the kernel to manipulate a file or a directory
- struct inode_operations
  - Describe the filesystem's implemented function that the VFS can invoke on an inode
  - e.g. `inode->i_op->truncate(inode)`

Some of inode Fields

```
struct inode {
    struct list_head i_sb_list;    /* inodes in the superblock */
    struct list_head i_dentry;     /* dentries referencing this inode*/
    unsigned long i_ino;           /* inode number */
    unsigned int i_nlink;          /* number of hard links */
    uid_t i_uid;                   /* user id of owner */
    gid_t i_gid;                   /* group id of owner */
    loff_t i_size;                 /* file size in bytes */
    struct timespec i_atime;       /* last access time */
    struct timespec i_mtime;       /* last modify time */
    struct timespec i_ctime;       /* last change time */
    umode_t i_mode;                /* access permissions */
    struct inode_operations *i_op; /* inode ops table */
    struct file_operations *i_fop; /* default inode ops */
    struct super_block *i_sb;      /* associated superblock */
    void *i_private;               /* fs private pointer */
...
};
```

Some of inode_operations Fields

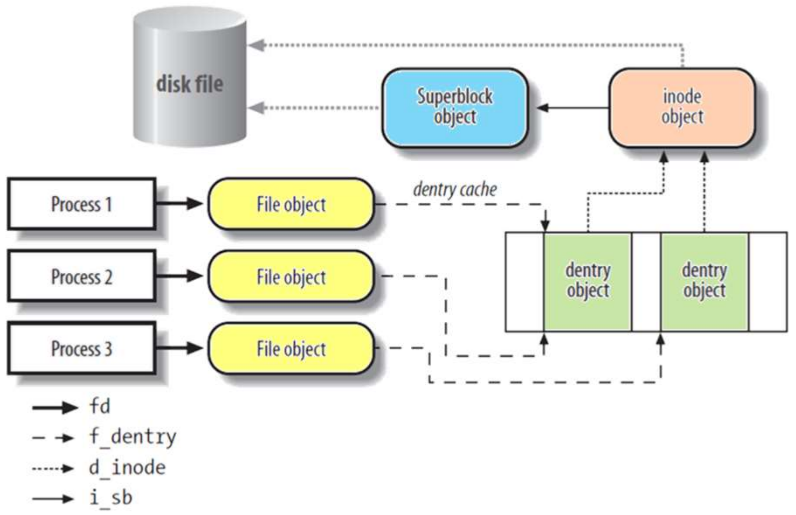
```
struct inode_operations {
    // create an inode
    int (*create) (struct inode *dir, struct dentry *dentry, int mode,
                   struct nameidata*);

    // searches a directory for an inode
    struct dentry *(*lookup)(struct inode *dir, struct dentry *dentry,
                             struct nameidata*);

    // link/unlink a hard link to a file
    int (*link) (struct dentry *old, struct inode *dir, struct dentry *new);
    int (*unlink) (struct inode *dir, struct dentry *dentry);

    // create an inode for a symbolic link
    int (*symlink) (struct inode *dir, struct dentry *dentry, const char *name);

    // create an inode for a new directory
    int (*mkdir) (struct inode *dir, struct dentry *dentry, int mode);

    // remove a directory
    int (*rmdir) (struct inode *dir, struct dentry *dentry);
...
};
```

Dentry Object

- struct dentry
  - A specific component in a path
    - E.g. `/`, `bin`, and `vi` are dentries in a path `/bin/vi`
  - dentry makes it easy to resolve a path and walk its components
- dentry state
  - Used: associated with an inode and the kernel is using it
  - Unused: associated with an inode, but the kernel is not using it (d_count is 0)
  - Negative: not associated with an inode
- dentry cache
  - After resolving each elements of a path, its dentries are cached at dcache

Some of dentry Fields

```
struct dentry {
    atomic_t d_count;                    /* usage count */
    int d_mounted;                       /* is this a mount point? */
    struct inode *d_inode;               /* associated inode */
    struct hlist_node d_hash;            /* list of hash table entries */
    struct dentry *d_parent;             /* dentry of parent directory*/
    struct qstr d_name;                  /* dentry name */
    struct list_head d_lru;              /* unused list */
    struct list_head d_subdirs;          /* subdirectories */
    struct dentry_operations *d_op;      /* dentry operations table */
    struct super_block *d_sb;            /* superblock of file */
...
};
```



Some of dentry_operations Fields

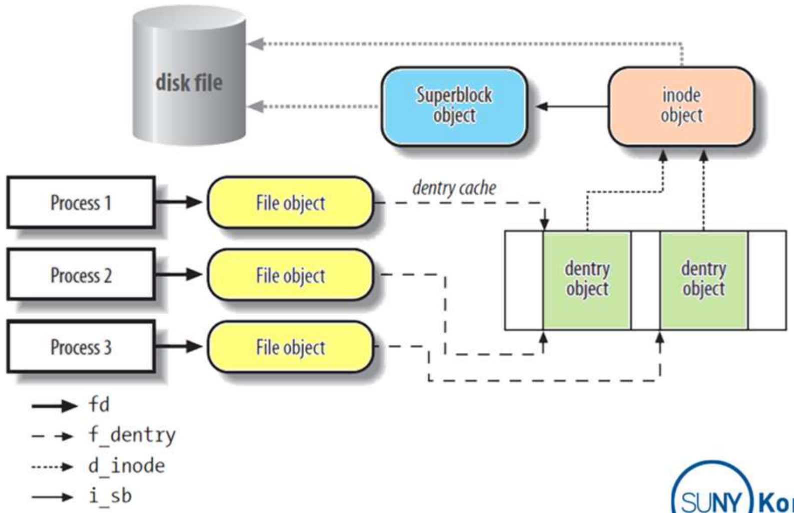
```
struct dentry_operations {
    // decide if dentry is still valid (network filesystem)
    int(*d_revalidate) (struct dentry *, struct nameidata *);

    // filesystem specific hash function
    int(*d_hash) (struct dentry *, struct qstr *);

    // compares two file names
    int(*d_compare) (struct dentry *, struct qstr *, struct qstr *);

    // called when d_count becomes 0
    int(*d_delete) (struct dentry *);

    // called when dentry is going to be freed
    void(*d_release) (struct dentry *);

    // called when dentry state becomes negative
    void(*d_iput) (struct dentry *, struct inode *);
...
};
```

myfs: Create an Inode

```c
static struct inode *myfs_create_inode(struct super_block *sb)
{
    struct inode *inode = new_inode(sb);
    PENTER;
    IF_NULL_GOTO(inode, fail);

//TODO: initialize i_ino, i_uid, i_gid, i_atime, i_mtime, i_ctime fields
//      using get_next_ino(), current_fsuid(), current_fsgid(), and
//      CURRENT_TIME

    return inode;
fail:
    return NULL;
}
```

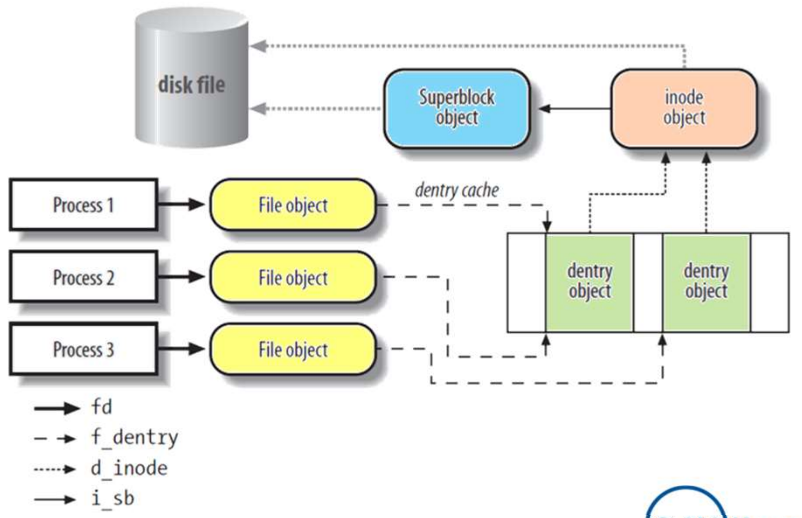
myfs: Create a Directory

```
static struct dentry *myfs_create_dir(struct super_block *sb,
    struct dentry *dir, char *name) {
    struct dentry *dentry = NULL;
    struct inode *inode = NULL;

    PENTER;
    IF_NULL_GOTO(dentry = d_alloc_name(dir, name), fail);
    IF_NULL_GOTO(inode = myfs_create_inode(sb), fail);
//TODO: update i_mode, i_size(= 64), i_op, and i_fop fields
//       using DEFAULT_MODE_DIR, simple_dir_inode_operations and
//       simple_dir_operations
    d_add(dentry, inode);
    return dentry;
fail:
    if(dentry)
        dput(dentry);
    if(inode)
        iput(inode);
    return NULL;
}
```



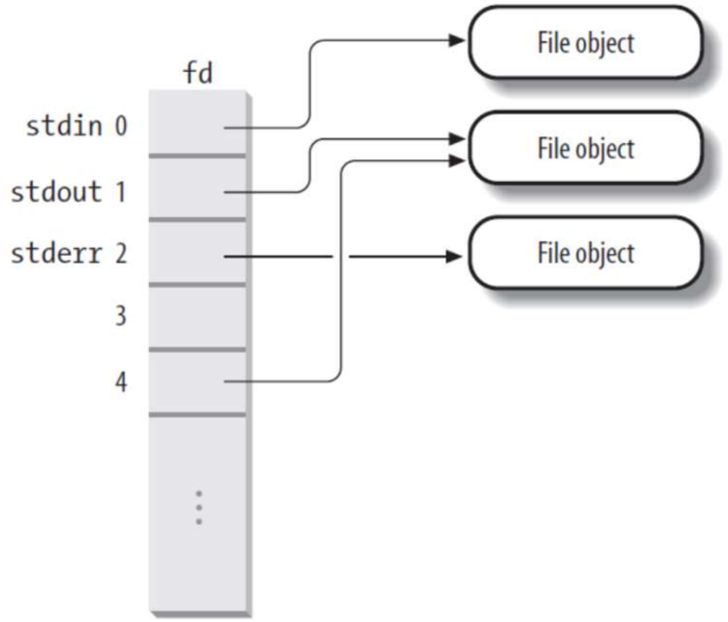
myfs: Create a File

```c
static struct dentry *myfs_create_file(struct super_block *sb,
    struct dentry *dir, char *name, char *msg) {
    struct dentry *dentry = NULL;
    struct inode *inode = NULL;

    PENTER;
    IF_NULL_GOTO(dentry = d_alloc_name(dir, name), fail);
    IF_NULL_GOTO(inode = myfs_create_inode(sb), fail);
//TODO: update i_mode(=DEFAULT_MODE_FILE), i_size(=strlen(msg)), and
//             i_fop(=&myfs_file_ops)
//      allocate MAX_FILE_SIZE buffer and make i_private pointing to it,
//      copy the contents of msg to i_private: use strcpy,
//      add inode to dentry: use d_add(dentry, inode)
    return dentry;
fail:
    if(dentry)
        dput(dentry);
    if(inode && inode->i_private)
        kfree(inode->i_private);
    if(inode)
        iput(inode);
    return NULL;
}
```



File Object

- struct file
  - Represents a file opened by a process
  - files_struct in task_struct has fd pointing to an array of file pointers.



- struct file_operations
  - Form the basis of the standard Unix system calls related to files

Some of file Fields

```
struct file {
    struct path f_path;                   /* contains the dentry */
    struct file_operations *f_op; /* file operations table */
    atomic_t f_count;                      /* file object’s usage count */
    unsigned int f_flags;                 /* flags specified on open */
    mode_t f_mode;                         /* file access mode */
    loff_t f_pos;                          /* file offset (file pointer) */
    void *private_data;                   /* tty driver hook */
...
};
```

Some of file_operations Fields

```c
struct file_operations {
    // update the file offset
    loff_t (*llseek) (struct file *, loff_t, int);

    // read the file from the offset
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);

    // write to file from the offset
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);

    // read the next directory
    int (*readdir) (struct file *, void *, filldir_t);

    // sleep until any activity occurs on the file
    unsigned int (*poll) (struct file *, struct poll_table_struct *);

    // memory map the file to the address space
    int (*mmap) (struct file *, struct vm_area_struct *);

    // create a new file linked to the inode object
    int (*open) (struct inode *, struct file *);
...
};
```

myfs: File Operations

```
static ssize_t myfs_read_file(struct file *file, char *buf,
    size_t count, loff_t *offset) {
    char *msg = //TODO: we will read from file’s private_data
    long msglen = strlen(msg);

    PENTER;
    return simple_read_from_buffer(buf, count, offset, msg, msglen);
}

static ssize_t myfs_write_file(struct file *file, const char *buf,
    size_t count, loff_t *offset) {
    char *msg = //TODO: we will write to file’s private_data
    ssize_t ret = 0;

    PENTER;
    ret = simple_write_to_buffer(msg, MAX_FILE_SIZE-1, offset, buf, count);
    if(ret >= 0) {
        file->f_inode->i_size = ret;
        msg[ret] = 0;
    }

    return ret;
}
```

myfs: File Operations

```
static int myfs_open(struct inode *inode, struct file *file) {
    PENTER;
    //TODO: make file’s private_data pointing to the buffer
    //      allocated when creating the file (inode->i_private)
    return 0;
}

static struct file_operations myfs_file_ops = {
    //TODO: initialize the open, read, and write fields
};

// To test myfs in user space
mkdir tmp  #if tmp doesn’t already exit
sudo mount –t myfs none ./tmp # mount the filesystem myfs
cd tmp
ls tmp
...
sudo umount ./tmp  # unmount the filesystem myfs
```

Assignment 6

- Build a filesystem named procfs
  - Its directory tree is generated from the process family tree (see assignment 2)
    - init_task is the root directory
    - for each child of a process, create a subdirectory named as name_pid, where name is from task->comm (non-alphanumeric characters are replaced with ‘_’) and pid is from task->pid

Assignment 6

- For each process create a file named as proc_info.txt whose contents are as below (use assignment 2)
  - process name: task->comm
  - pid: task->pid
  - tty name: task->signal->tty->name (or NA if not available)
- There is no write operation for the file and set its default mode to read-only for user, group, and others

- Use 0xFEEDBEEF for the filesystem's magic number
- Due date: TBD

Assignment 6

- For this assignment, implement the main functions and helper functions below

```
// main functions
static struct dentry *procfs_create_file(struct super_block *sb,
                          struct dentry *dir, struct task_struct *task);
static struct dentry *procfs_create_dir(struct super_block *sb,
                          struct dentry *dir, struct task_struct *task);
static int procfs_create_tree(struct super_block *sb,
                          struct dentry *dir, struct task_struct *task);

// helper functions
static int task_to_contents(struct task_struct *task, char *buf, size_t buf_size);
static int task_to_name(struct task_struct *task, char *buf, size_t buf_size);

struct task_frame {
    struct task_struct *task;
    struct dentry *dir;
};
static struct task_frame frame_stack[1000];
static int frame_sp = 0;
static void push_frame(struct task_struct *task, struct dentry *dir);
static void pop_frame(struct task_struct **task, struct dentry **dir);
```

Assignment 6 (Example Result)

```
ykwon4@youngbox2:~/home$ sudo mount -t procfs none ./tmp
ykwon4@youngbox2:~/home$ mount
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
devtmpfs on /dev type devtmpfs (rw,relatime,size=1022864k,...
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,...
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
...
none on /home/ykwon4/home/tmp type procfs (rw,relatime)
```

```
ykwon4@youngbox2:~/home$ ls -alR tmp
tmp:
total 4
drwxr-xr-x 2 root   root      0 11월 27 10:29 .
drwxrwxr-x 3 ykwon4 ykwon4 4096 11월 26 12:19 ..
dr-xr-xr-x 1 root   root     64 11월 27 10:29 kthreadd_2
-r--r--r-- 1 root   root     64 11월 27 10:29 proc_info.txt
dr-xr-xr-x 1 root   root     64 11월 27 10:29 systemd_1

tmp/kthreadd_2:
total 0
dr-xr-xr-x 1 root root 64 11월 27 10:29 .
drwxr-xr-x 2 root root  0 11월 27 10:29 ..
dr-xr-xr-x 1 root root 64 11월 27 10:29 acpi_thermal_pm_759
dr-xr-xr-x 1 root root 64 11월 27 10:29 ata_sff_480
dr-xr-xr-x 1 root root 64 11월 27 10:29 bioset_398
...
tmp/kthreadd_2/acpi_thermal_pm_759:
...

ykwon4@youngbox2:~/home$ cat tmp/proc_info.txt
name: swapper/0
pid: 0
tty name: NA
```

```
ykwon4@youngbox2:~/home$ sudo umount ./tmp
ykwon4@youngbox2:~/home$ mount
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
devtmpfs on /dev type devtmpfs (rw,relatime,size=1022864k,...
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
selinuxfs on /sys/fs/selinux type selinuxfs (rw,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,...
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
...

# procfs should be unmounted
```

Thank you for your attention during the semester!

Any questions or comments?

- Please submit your Course Evaluation at https://p22.courseval.net/etw/ets/et.asp?nxappid=SU2&nxmid=start