

## Directory permissions

- r (read): Can read directory entries.
  - ls command can list the directory.
  - readdir() can get directory entry.
- w (write): Can add new directory entry, modify existing directory entry, or delete directory entry.
  - new directory entry: when create new sub-directory (e.g. mkdir) or file (e.g. touch, ...).
  - modify existing directory entry: rename file or sub-directory.
  - delete directory entry: delete file or sub-directory (e.g. rm, rmdir, ...)
- x (execute): Can browse the directory
  - cd command (chdir() syscall) will work

## Current Directory

- Each process has a current directory.
- The relative paths in the program are processed w.r.t. the current directory of that process.
- In Linux, struct task\_struct (PCB) has a member struct fs\_struct.

```
struct fs_struct {  
    struct dentry *pwd;  
    struct dentry *root;  
    // ...  
};
```

- By default current directory is inherited from the parent process.

## chdir(dirpath)

- Change the current directory of the current process to the given directory path.
- arg1: dir path
- returns: 0 on success.

## File IO syscalls

- open()
- read()
- write()
- close()
- lseek()

## open() syscall

- fd = open("file-path", flags, mode);
  - arg1: path of file to be opened
  - arg2: flags - how you want to open the file
    - O\_RDONLY, O\_WRONLY, O\_RDWR -- read-write flags
    - O\_TRUNC -- delete the contents of file while opening

- O\_APPEND -- write at the end of file
- O\_CREAT -- create a new file (if not present) -- must give arg3
- arg3: mode - permissions for new file - octal number
- returns file descriptor on success, and -1 on failure.
  - file descriptor is int that uniquely identifies the file in that process.
  - fd is used in other file io syscalls e.g. close(), read(), write(), lseek().

## close() syscall

- int close(fd)
  - arg1: fd returned by open().
  - Returns 0 on success and -1 on error
- Decrement ref count in open file table entry (struct file).
- If ref count drops to zero, OFT entry is deleted (from OFT).

## read() syscall

- count = read(fd, buf, length);
  - arg1: file descriptor
  - arg2: buffer in which you want to read
  - arg3: length of buffer
  - Returns: Number of characters read

## write() syscall

- ssize\_t write(int fd, const void \*buf, size\_t count);
  - arg1: file descriptor
  - arg2: buffer from which you want to write
  - arg3: length of data
  - Returns: Number of characters written

## lseek() syscall

- Change the file position in open file table entry (struct file --> f\_pos).
- And returns new file position (from the beginning of the file).
- off\_t lseek(int fd, off\_t offset, int whence);
- Examples:
  - lseek(fd, 0, SEEK\_SET);
    - filp->f\_pos = 0;
  - lseek(fd, offset, SEEK\_SET);
    - filp->f\_pos = offset;
  - lseek(fd, 0, SEEK\_END);
    - filp->f\_pos = size; // file size (from the inode)
  - lseek(fd, offset, SEEK\_END);
    - filp->f\_pos = size + offset; // note that offset will be negative
  - lseek(fd, offset, SEEK\_CUR);
    - filp->f\_pos = filp->f\_pos + offset;

# FileSystem Operations

- Directory Operations
  - Create directory -- mkdir()
  - Delete directory -- rmdir()
  - List directory -- opendir(), readdir(), closedir()
  - Change directory -- chdir()
- File Operations
  - Rename -- link() + unlink()
  - Delete -- unlink()
  - Links -- link(), symlink()
  - Get Info -- stat()
  - Change permissions -- chmod()
  - Change owner -- chown()

## rm command

- The rm command in Linux, internally calls unlink() system call.
- `int unlink(const char *filepath);`

### unlink() syscall

- It deletes directory entry of the file.
- It decrements link count in the inode by 1.
- If link count = 0, the inode is considered to be deleted/free (updated into super-block). It can be reused for any new file.
- When inode is marked free, data blocks are also made free, so that they can also be reused for some new file.

## Links

- Links are shortcuts to access deeply located files quickly.
- There are two types of links in Linux/UNIX.
  - 1. Symbolic Link
  - 2. Hard Link

## Symbolic Link

- `terminal> ln -s /path/of/target/file linkpath`
- Internally use symlink() syscall.
  - `man symlink`
  - `int symlink(const char *target_path, const char *link_path);`

### symlink() syscall

- A new link file is created (new inode and new data block is allocated), which contains info about the target file (absolute or relative path).
- Link count is not incremented.
- If target file is deleted, the link becomes useless.

- Can create symlinks for directories also.

## Hard Link

- `terminal> ln targetfilepath linkfilepath`
- Internally use `link()` syscall.
  - `man link`
  - `int link(const char *target_path, const char *link_path);`

### `link()` syscall

- A new directory entry is created, which has a new name and same inode number. No new file (inode and data blocks) is created.
- Link count in the inode of the file is incremented.
- If directory entry of target file is deleted (`rm` command), file can be still accessed by link directory entry.
- Cannot create hard link for directories, because it may lead to infinite recursion (while traversing directories recursively e.g. `ls -R`)

## Reading

- \* Galvin slides (File System & IO subsystem)
- \* Professional Linux Kernel Architecture (Virtual File System, Extended File System)
- \* Beginning Linux Programming (File SysCalls Programming)

## Assignments

1. Input dir path from command line and display dir contents.
  - `terminal> ./assign01.out /home`
2. Input dir path from command line and display dir contents recursively. (Optional)
  - `terminal> ./assign01.out /tmp`
  - Hint: if(`ent->d_type == DT_DIR`) `recursive_dir_list(ent->d_name)`