File System Mounting

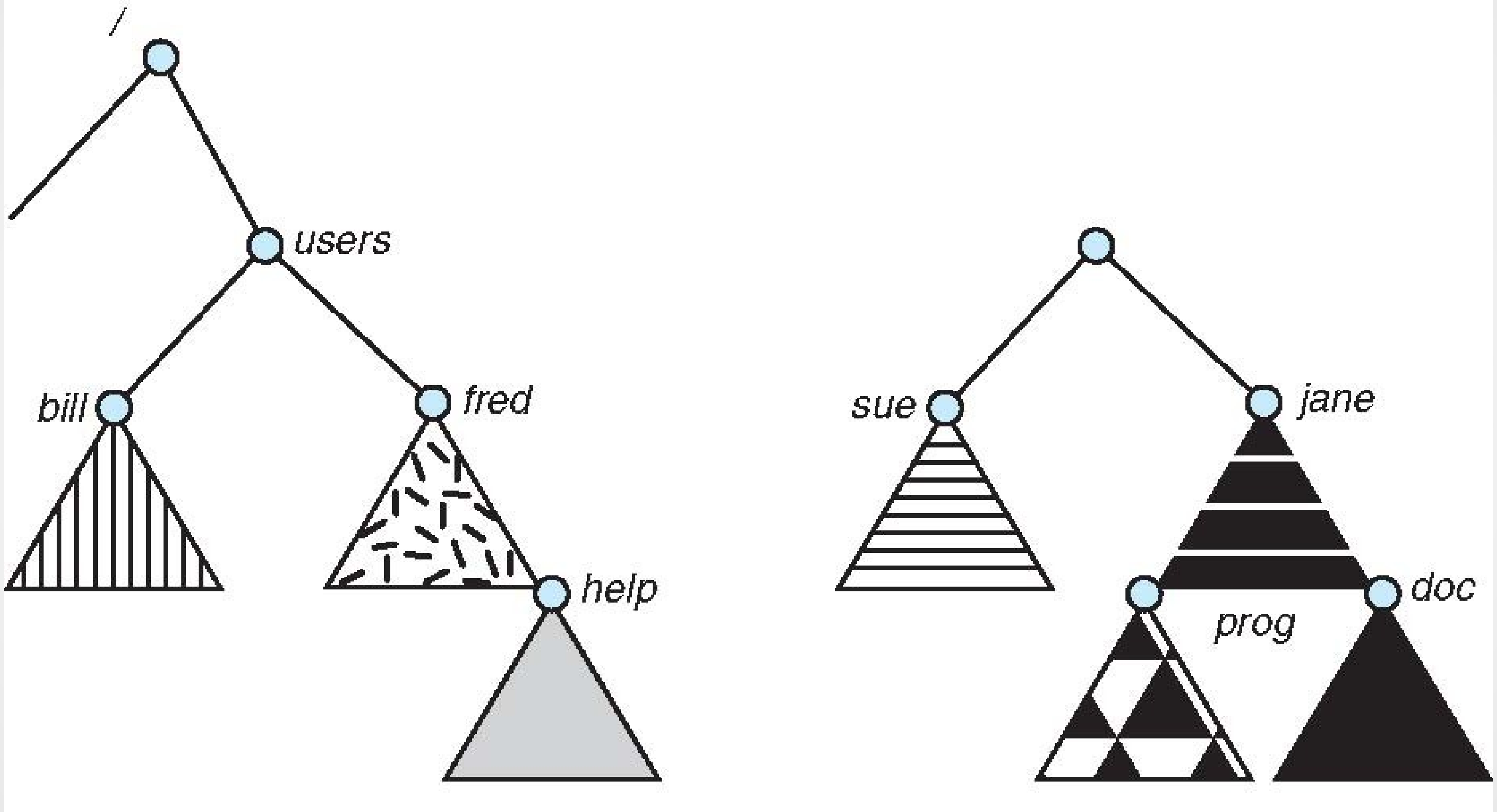
- A file system must be **mounted** before it can be accessed
  - Privileged operation
  - First check for valid file system on volume
  - Kernel data structure to track mount points
- Some systems have separate designation for mount point (i.e. “c:”)
- Others integrate mounted file systems into existing directory naming system
  - In separate space (i.e. /volumes) or within current name space

File System Mounting

- A unmounted file system on */device/dsk* (i.e., Fig. 11-11(b)) is mounted at a **mount point**

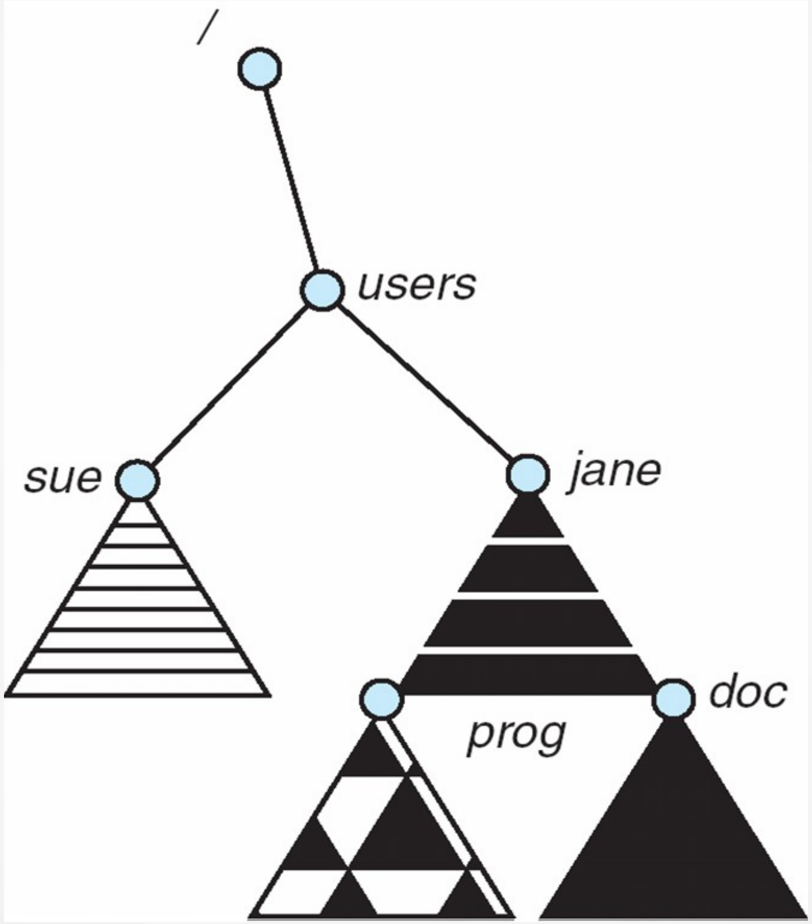
  What if the mount point already has contents?
- Configuration file or data structure to track default mounts
  - Used at reboot or to reset mounts
- What if files are open on a device that is being **unmounted**?

(a) Existing (b) Unmounted Partition



(a) (b)

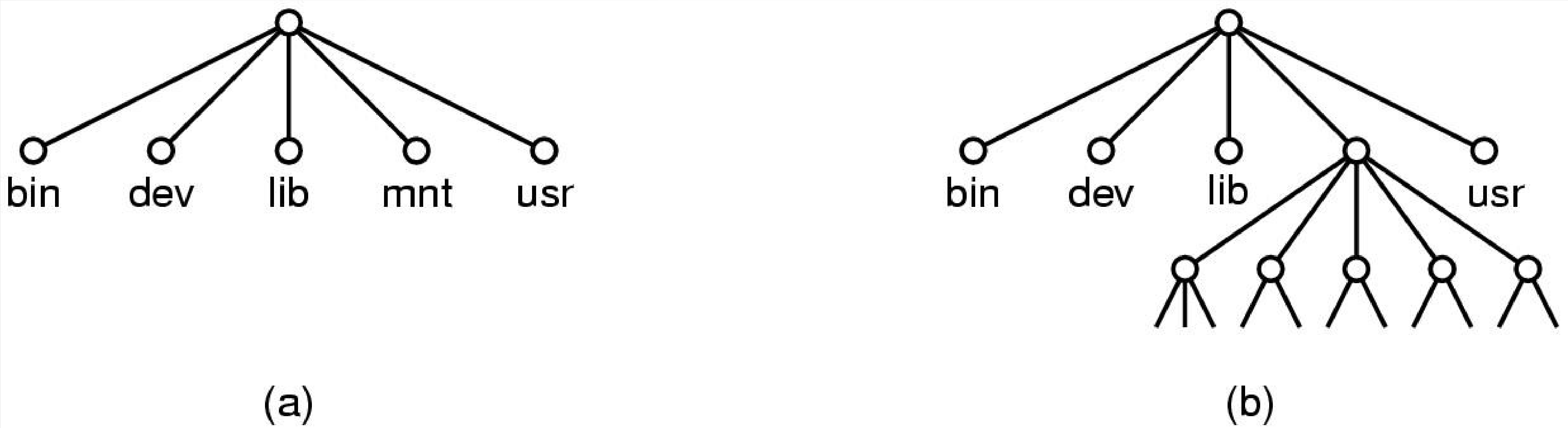
Mount Point



File System Mounting

- Mount allows two FSes to be merged into one
  - For example you insert your floppy into the root FS

```
mount("/dev/fd0", "/mnt", 0)
```



Remote file system mounting

- Same idea, but file system is actually on some other machine
- Implementation uses remote procedure call
  - Package up the user’s file system operation
  - Send it to the remote machine where it gets executed like a local request
  - Send back the answer
- Very common in modern systems

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method

Path Names

- To access a file, the user should either:
  - Go to the directory where file resides, or
  - Specify the **path** where the file is
- Path names are either absolute or relative
  - Absolute: path of file from the root directory
  - Relative: path from the current working directory
- Most OSes have two special entries in each directory:
  - "." for current directory and ".." for parent

Directory Organization – Hierarchical

- Most systems support idea of current (working) directory
  - Absolute names – fully qualified from root of file system
    - `/usr/group/foo.c, ~/kernelSrc/config.h`
  - Relative names – specified with respect to working directory
    - `foo.c, bar/bar2.h`
  - A special name – the working directory itself
    - “.”
- Modified Hierarchical – Acyclic Graph (no loops) and General Graph
  - Allow directories and files to have multiple names
  - Links are file names (directory entries) that point to existing (source) files

Links

- *Symbolic (soft) links:* uni-directional relationship between a file name and the file
  - Directory entry contains *text* describing *absolute* or *relative* path name of original file
  - If the source file is deleted, the link exists but pointer is invalid
- *Hard links:* bi-directional relationship between file names and file
  - A *hard link* is directory entry that points to a source file's metadata
  - Metadata maintains *reference count* of the number of hard links pointing to it – *link reference count*
  - Link reference count is decremented when a hard link is deleted
  - File data is deleted and space freed when the link reference count goes to zero

Unix-Linux Hard Links

- File may have more than one *name* or *path*
- **`rm, mv`** —*directory* operations, not *file* operations!
  - The *real* name of a Unix file is internal name of its metadata
    - Known only to OS!
- Hard links are not used very often in modern Unix practice
  - *Exception:* Linked copies of large directory trees!
  - (Usually) safe to regard last element of path as *name* of file

Directory Operations

- *Create:*
  - Make a new directory
- *Add, Delete entry:*
  - Invoked by file create & destroy, directory create & destroy
- *Find, List:*
  - Search or enumerate directory entries
- *Rename:*
  - Change name of an entry without changing anything else about it
- *Link, Unlink:*
  - Add or remove entry pointing to another entry elsewhere
  - Introduces possibility of loops in directory graph
- *Destroy:*
  - Removes directory; *must be empty*

Directories (continued)

- *Orphan:* a file not named in any directory
    - Cannot be opened by *any* application (or even OS)
    - May not even have name!
- Tools
    - FSCK – check & repair file system, find orphans
    - *Delete_on_close* attribute (in metadata)
- Special directory entry: “..” ⇒ parent in hierarchy
    - Essential for maintaining integrity of directory system
    - Useful for relative naming

Directories — Summary

- Fundamental mechanism for interpreting file names in an operating system
- Widely used by system, applications, and users

File Access Rights

- Types of Users:
  - Owner/user (u)
  - Group (g)
  - All/Other (o)
- Types of Permissions:
  - Read (r)
  - Write (w)
  - Execute (x)
- Types of Files
  - Directories
  - Other files

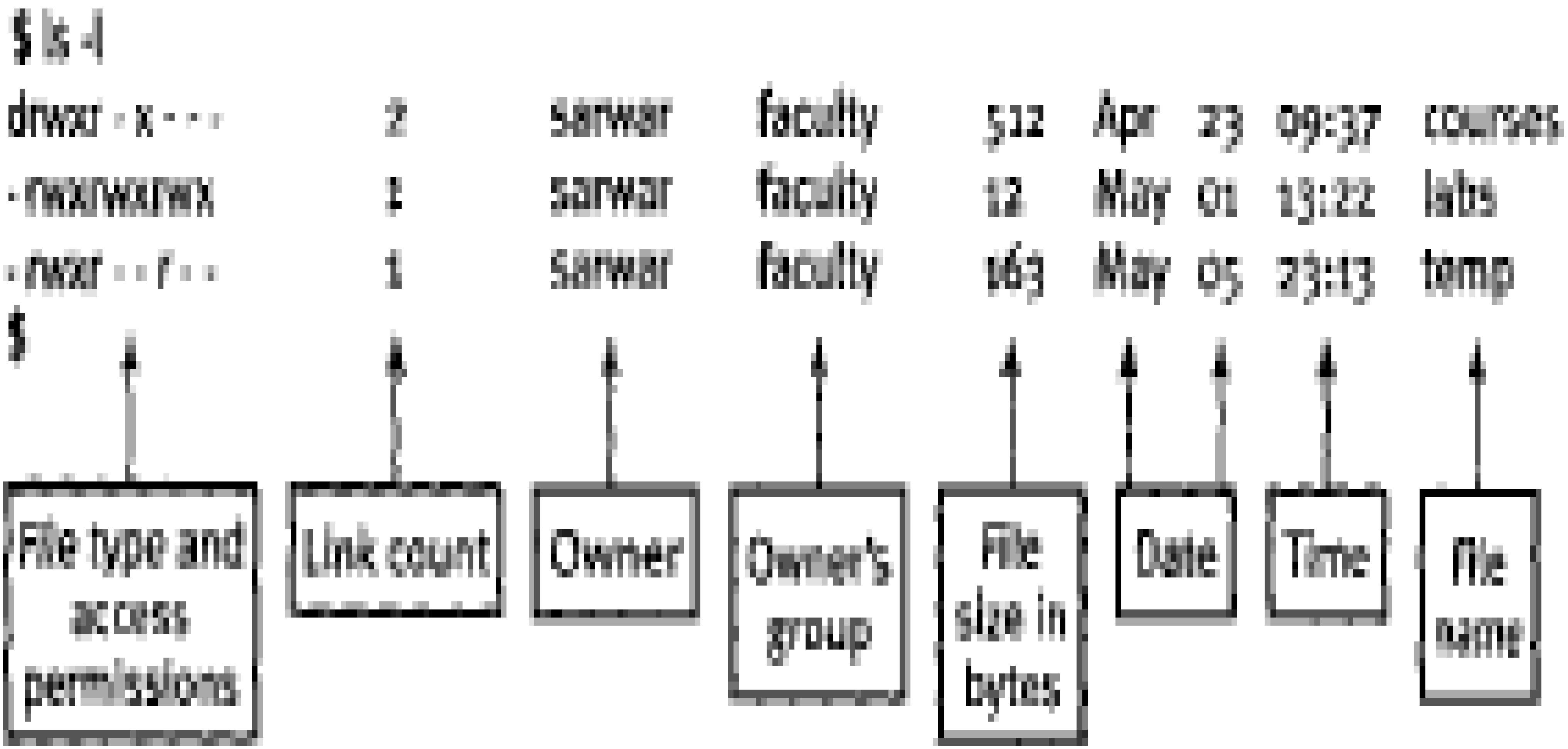
Directory Permissions

read = list files in the directory

write = add new files to the directory

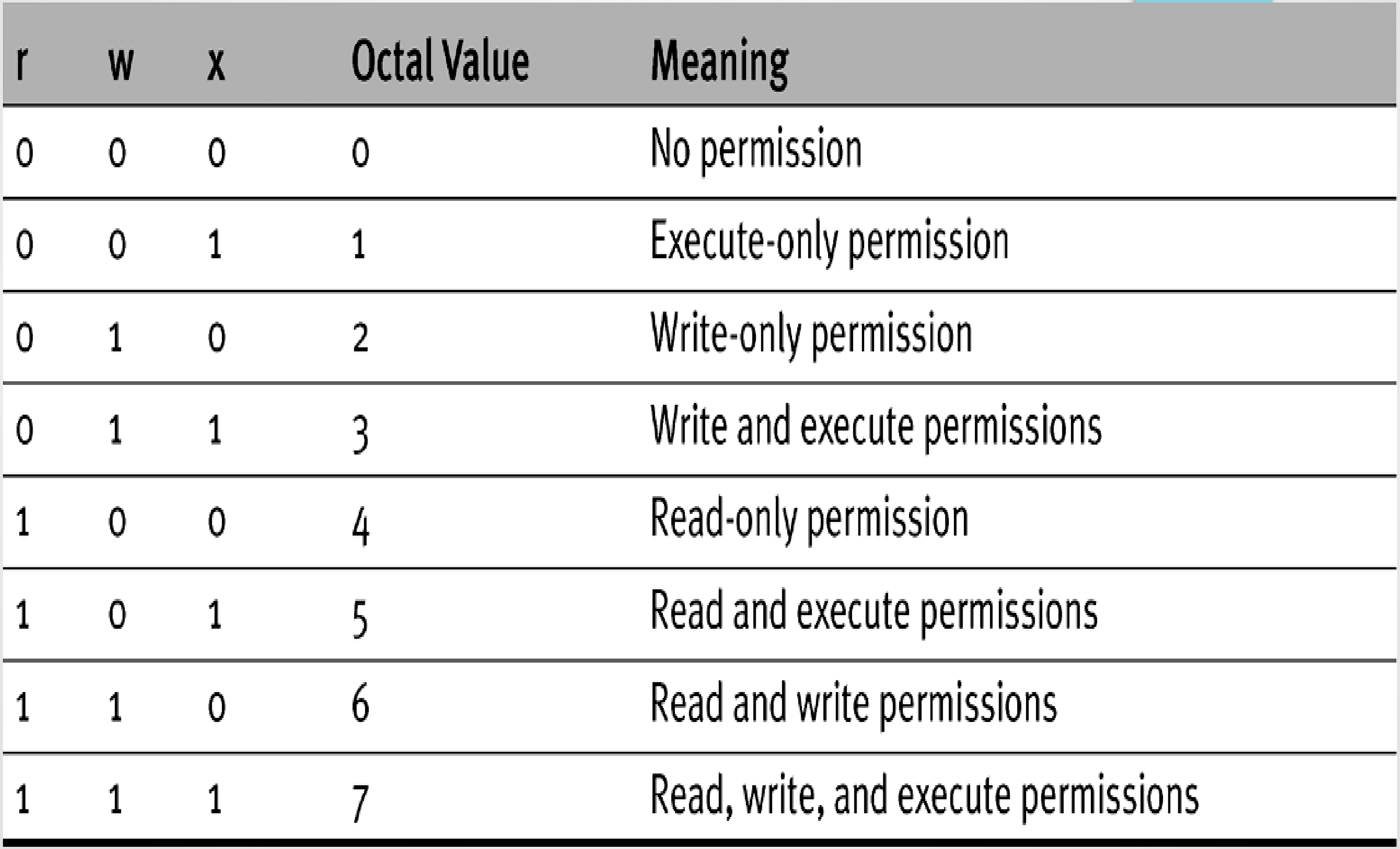
execute = access files in the directory

Determining File Access Rights



```
$ ls -l
drwxr-x---    2   sarwar   faculty    512  Apr  23  09:37  courses
-rwxrwxrwx    1   sarwar   faculty    12   May  01  13:22  labs
-rwxr--r--    1   sarwar   faculty    163  May  05  23:13  temp
$
```

Permission Values



r	w	x	Octal Value	Meaning
0	0	0	0	No permission
0	0	1	1	Execute-only permission
0	1	0	2	Write-only permission
0	1	1	3	Write and execute permissions
1	0	0	4	Read-only permission
1	0	1	5	Read and execute permissions
1	1	0	6	Read and write permissions
1	1	1	7	Read, write, and execute permissions