File System Calls – Part I

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- None
  - Sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program

File Attributes

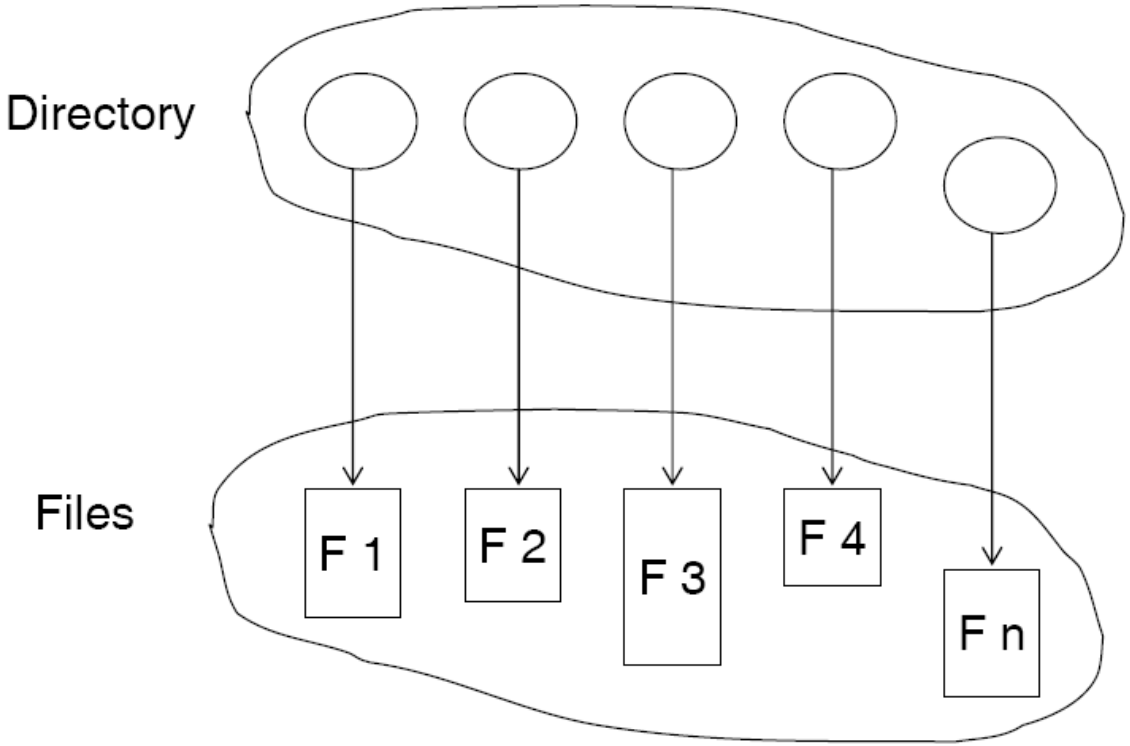
- **Name**–only information kept in human-readable form.
- **Type**–needed for systems that support different types.
- **Location**–pointer to file location on device.
- **Size**–current file size.
- **Protection**–controls who can do reading, writing, executing.
- **Time**, **date**, **and user identification**–data for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk.

File Operations

- create
- write
- read
- reposition within file
  - file seek
- delete
- truncate
- open(Fi)
  - search the directory structure on disk for entry Fi, and move the content of entry to memory.
- close (Fi)
  - move the content of entry Fi in memory to directory structure on disk.

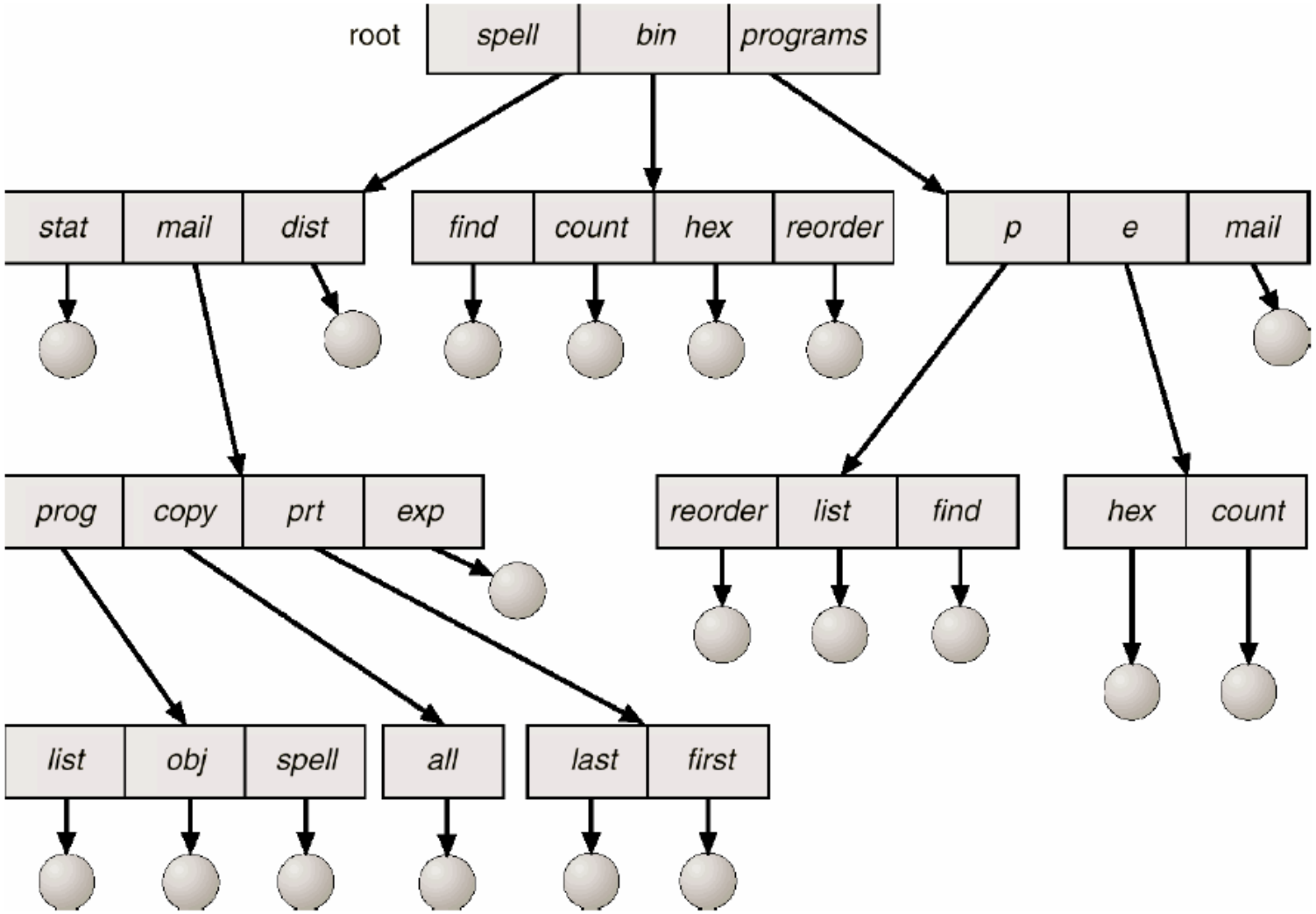
Directory Structure

- A collection of nodes containing information about all files.



- Both the directory structure and the files reside on disk.

Tree-Structured Directories (1)



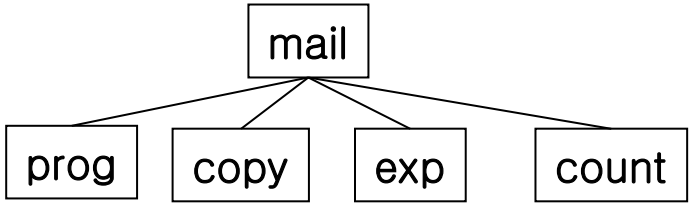
Tree-Structured Directories (2)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list

```
[Eunui-MacBook-Pro-10:~ hasoo$ type test
 test is a shell builtin
[Eunui-MacBook-Pro-10:~ hasoo$ type cp
 cp is /bin/cp
[Eunui-MacBook-Pro-10:~ hasoo$ type cd
 cd is a shell builtin
[Eunui-MacBook-Pro-10:~ hasoo$ type find
 find is /usr/bin/find
```

Tree-Structured Directories (3)

- Absolute or relative path name
- Creating a new file is done in current directory.
- Delete a file
    - **rm**<file-name>
- Creating a new subdirectory is done in current directory.
    - **mkdir** <dir-name>
  - Example: if in current directory /spell/mail
    - **mkdir** <dir-name>



- Deleting "mail"⇒deleting the entire subtree rooted by "mail".

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

	R	W	X
– a) owner access 7⇒	1	1	1
– b) groups access 6⇒	1	1	0
– c) public access 1⇒	0	0	1

- For a particular file (say game) or subdirectory, define an appropriate access.
  - **chmod** 761 game

```
[Eunui-MacBook-Pro-10:~ hasoo$ ls -la | grep result
 -rw-r--r--   1 hasoo  staff    255  3  9 05:55 result
[Eunui-MacBook-Pro-10:~ hasoo$ chmod go+x result
[Eunui-MacBook-Pro-10:~ hasoo$ ls -la | grep result
 -rw-r-xr-x   1 hasoo  staff    255  3  9 05:55 result
```

File Attributes - details

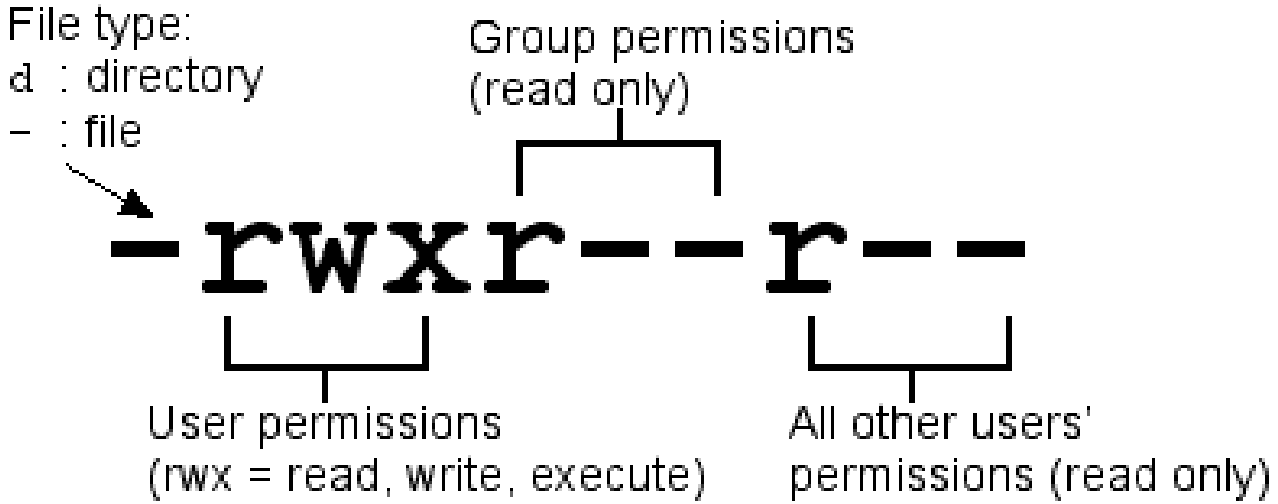
Attribute	Value meaning
File type	Type of file
Access permission	File access permissions for different users
Hard link count	Number of hard links of a file
UID	User ID of file owner
GID	Group ID of file
File size	File size in bytes
Last access time	Time the file was last accessed
Last modification time	Time the file was last modified
Last change time	Time the file attribute was last changed
Inode number	Inode number of the file
File system ID	File system ID where the file is stored

File Types - details

- Regular (-)
- Directory (d)
- Block device file (b)
- character device file (c)
- Domain socket (s)
- Pipe (FIFO file) (p)
- Symbolic link (l)

File Access Permissions (1)

- 3 different categories
  - *r w x*
- 3 different types of users
  - *u g o*



File Access Permissions (2)

- Directory
  - Whenever want to open any type of file by name, must have execute permission in each directory mentioned in the name.
  - Read permission for directory
    - Lets us read the directory, obtaining a list of all the filenames in the directory.
  - Write permission on directory
    - Lets us create new files and remove existing files in that directory
  - Execution permission for directory (search permission)
    - Lets us pass through the directory when it is a component of a pathname that we are trying to access.

Extra Permissions for Executable File (1)

- User ID
  - Real
    - Identifies the user who is responsible for the running process.
  - Effective
    - Used to assign ownership of newly created files, to check file access permissions, and to check permission to send signals to processes.
    - To change euid: executes a *setuid-program* that has the setuid bit set or invokes the setuid system call.
      - setuid(*uid*) system call:
    - Typical program that calls the setuid(uid) system call
      - passwd, login, mkdir, etc.
  - Real and effective uid: inherit (fork), maintain (exec).
- Group ID
  - Real, effective

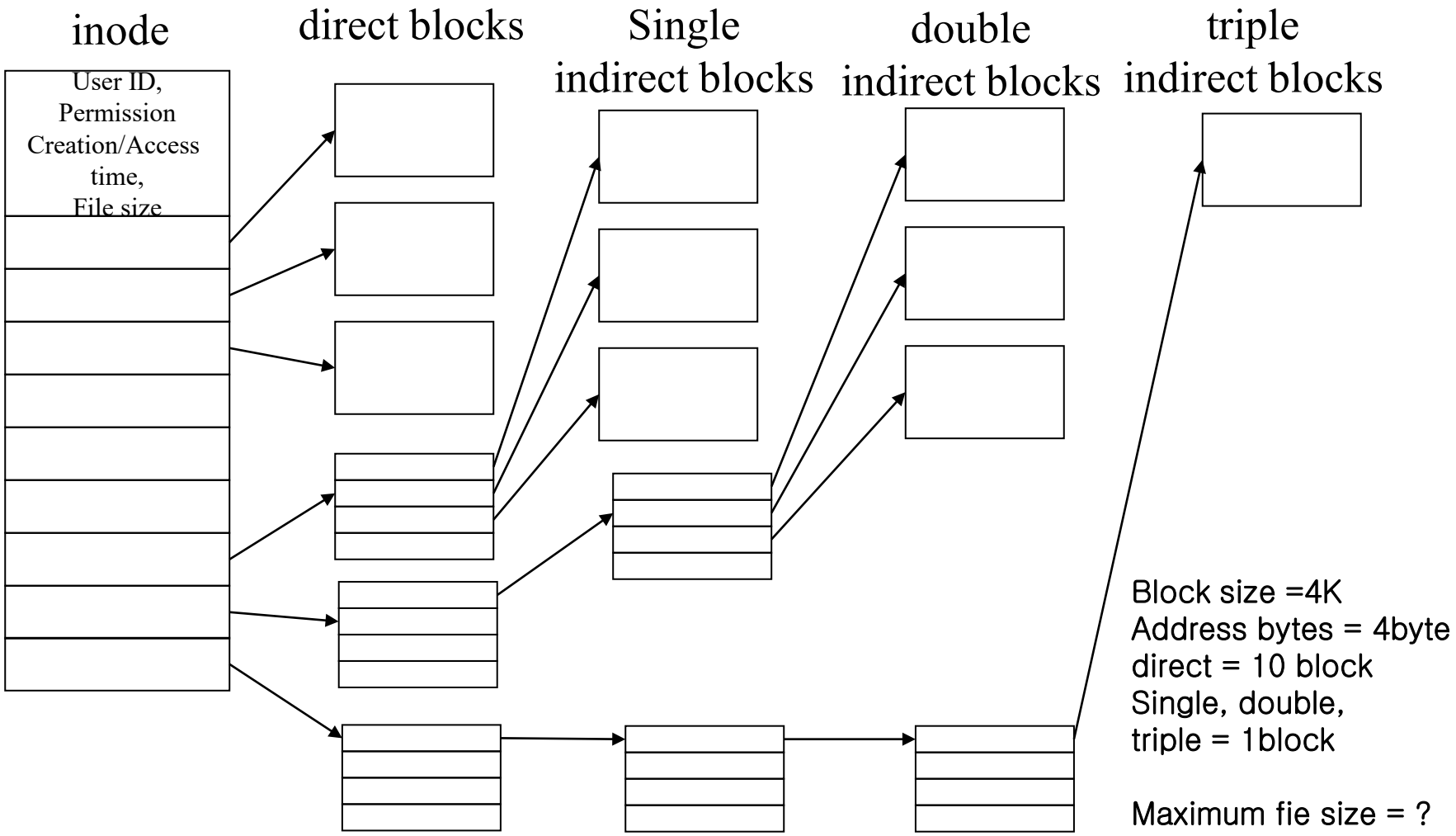
Extra Permissions for Executable File (2)

- Some special run-time properties
  - Set-user-id bit: 04000
    - When the file is started, its effective user ID is taken from the *file owner*.
  - Set-group-id bit: 02000
    - Does the same thing for the file's group ID.
  - Sticky bit (*save-text-image* permission): 01000
    - Designed to save time in accessing commonly used programs.
    - When it is executed, its program-text part will remain in the system's *swap area* until the system is halted. When the program next invoked, simply swaps it into memory (no directory search is needed)

Inodes

- The administrative information about a file is kept in a structure known as an *inode*.
  - Inodes in a file system, in general, are structured as an array known as an *inode table*.
- An inode number, which is an index to the inode table, *uniquely identifies* a file in a file system.

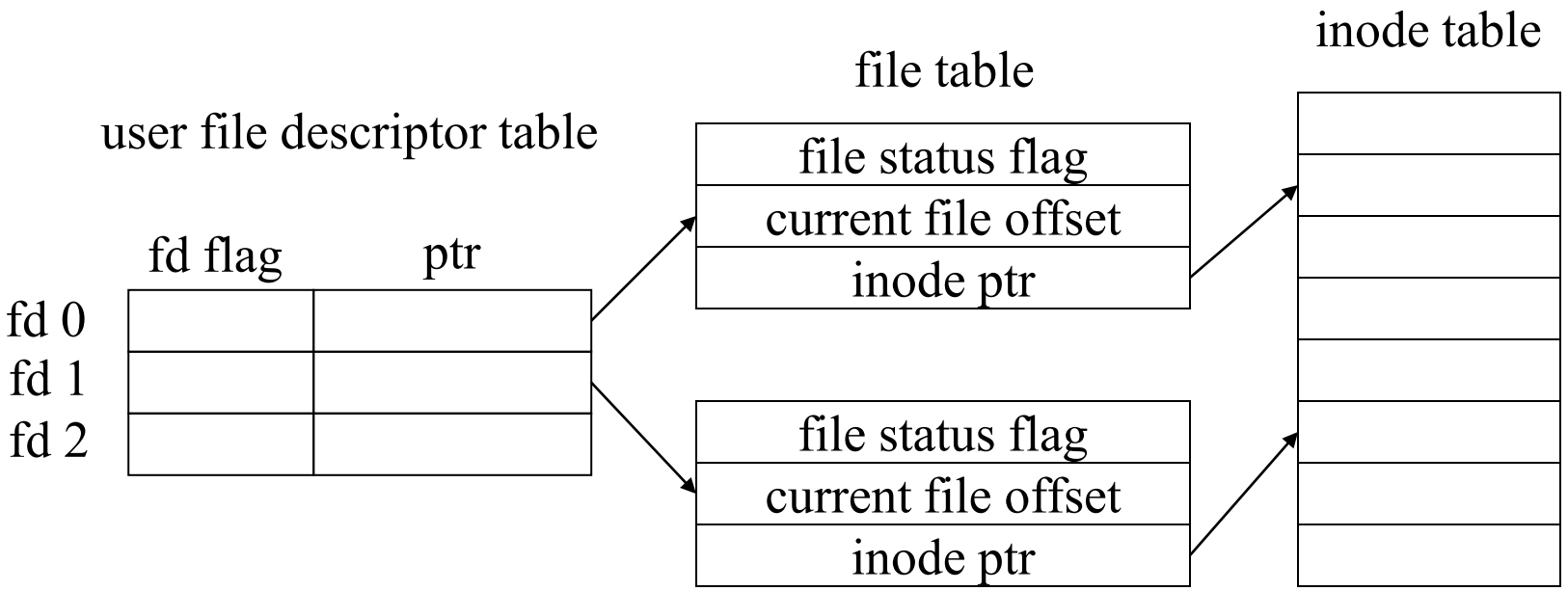
Inode and Data blocks



UNIX Kernel Support for Files

- Three important run-time tables
  - File descriptor table
  - File table
  - Inode table

Open File Tables



File Descriptor

- All open files are referred to by file descriptors.
- When we open an existing file or create a new file, the kernel returns a file descriptor to the process.
- To identify the file with the file descriptor.
- Predefined file descriptors in <unistd.h>.
  - STDIN_FILENO (0)
  - STDOUT_FILENO (1)
  - STDERR_FILENO (2)

Hard and Symbolic Links

Hard link	Symbolic link
Does not create a new inode	Creates a new inode
Increases hard link count	Not affecting the target file
Not for directories (except for root)	Can link to any type of files
Not allowed to go across file systems	Can link to any file anywhere