# UNIX FILES

## UNIX / POSIX file Types

The different type's files available in UNIX / POSIX are:

- Regular files Example: All .exe files, C, C++, PDF Document files.
- Directory files Example: Folders in Windows.
- Device files
    - Block Device files: A physical device that transmits block of data at a time.
      For example: floppy devices CDROMs, hard disks.
    - Character Device files: A physical device that transmits data in a character based manner.
      For example: Line printers, modems etc.
- FIFO files Example: PIPEs.
- Link Files

## Hard Links

It is a UNIX path or file name, by default files are having only one hard link

## Symbolic Links

Symbolic links are called soft links. Soft link are created in the same manner as hard links, but it requires –s option to the *ln* command. Symbolic links are just like shortcuts in windows.

## Differences between Hard links and Symbolic Links

| Hard Link | Soft Links |
|---|---|
| 1. Do not create new inode. | 1. Create a new inode. |
| 2. Cannot link directories unless super user privileges. | 2. Can link directories. |
| 3. Cannot link file across file systems. | 3. Can link files across file systems. |
| 4. Increase the hard link count. | 4. Does not change the hard link count. |
| 5. Always refer to the old file only, | 5. Always reference to the latest |

| | |
|---|---|
| means hard links can be broken by removal of one or more links. | version of the files to which they link. |

**UNIX Kernel supports for file / Kernel Data structure for file manipulation**

If open call succeeds, kernel establish the path between preprocess table to inode table through file table

The Steps involved in this process are:

**Step 1:** The kernel will search the process file descriptor table and look for first unused entry, if an entry is found, that entry will be designated to reference the file.

**Step 2:**The kernel scan the file table in its kernel space to find an unused entry that can be assigned to reference the file.

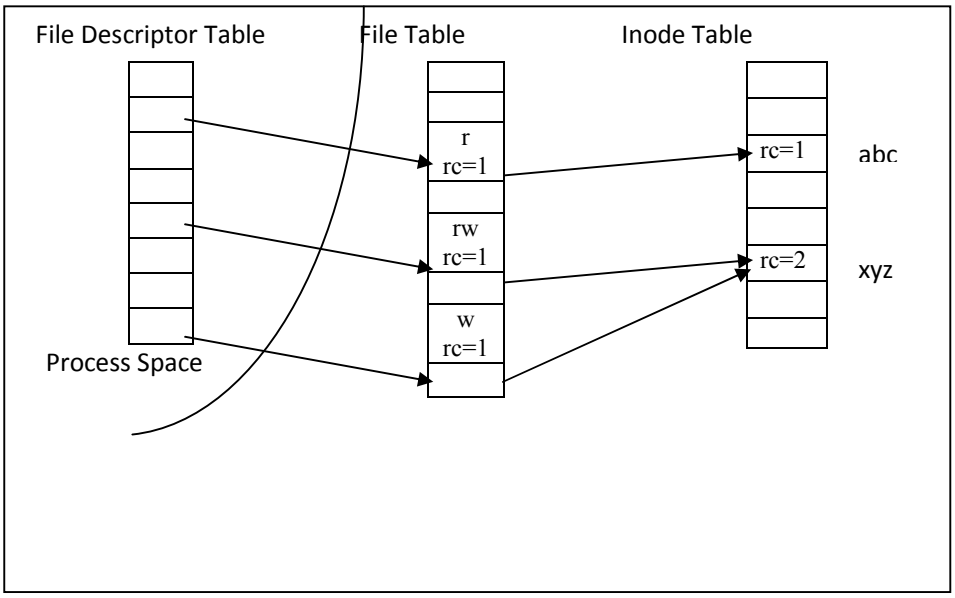
If an unused entry is found, the following events will occur.

The process's file table entry will be set to point to this file table entry.

- The file table entry will be set to point to the inode table entry where the inode record of the file is stored.
- The file table entry will contain the current file pointer of the open file.
- The file table entry will contain open mode that specifies that the file is open for read-only, write-only or read-write etc.
- The reference count in the file table entry is set to 1. The reference count keeps track of how many file descriptors from any process are referencing the entry.
- The reference count of the in-memory inode of the file is increased by 1. This count specifies how many file table entries are pointing to that inode.

If either step1 or step2 fails, the open function will return with a -1 failure status, no file descriptor table or file table entry will be allocated.

The figure shows a process's file descriptor table, the kernel file table and the inode after the process has opened three files: *abc* for read only, and *xyz* for read- write and *xyz* again for write only.




The reference count of an allocated file table entry is usually 1, but a process may

When a process calls the function *close* to close an opened file, the following sequence of events will occur.

1) The kernel sets the corresponding file descriptor table entry to be unused.
2) It decrements the reference count in the corresponding file table entry by 1. If the reference count is still non-zero, go to step 6.
3) The file table entry is marked as unused.
4) The reference count in the corresponding file inode table entry is set decremented by one. If the count is still non-zero go to step 6.
5) If the hard link count of the inode is not zero, it returns to the caller with a success status otherwise, it marks the inode table entry as unused and de- allocates all the physical disk storage of the file.
6) It returns to the caller to the process with 0 (success) statuses.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-vi-unix-system-programming-10cs62-notes.pdf