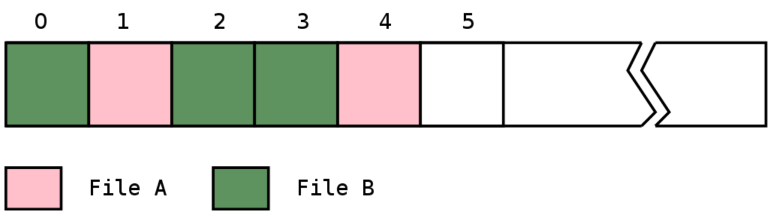
Filesystem Internals

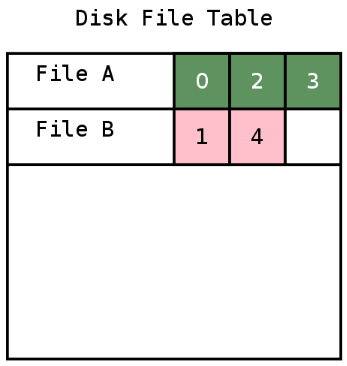
Zilogic Systems

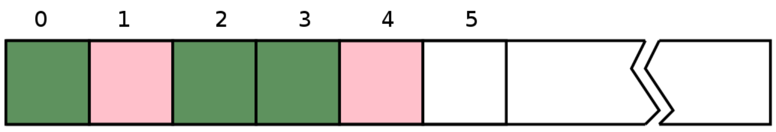
1. Need for a Filesystem

- Data corresponding to a file are stored in sectors of the harddisk.



- User has to remember the sectors that store the contents of a file.
- The user also has to keep track of which sectors are free.
- Managing files becomes tedious.
- The filesystem maintains a table, that maps filenames to the sectors that contain the contents of the file.

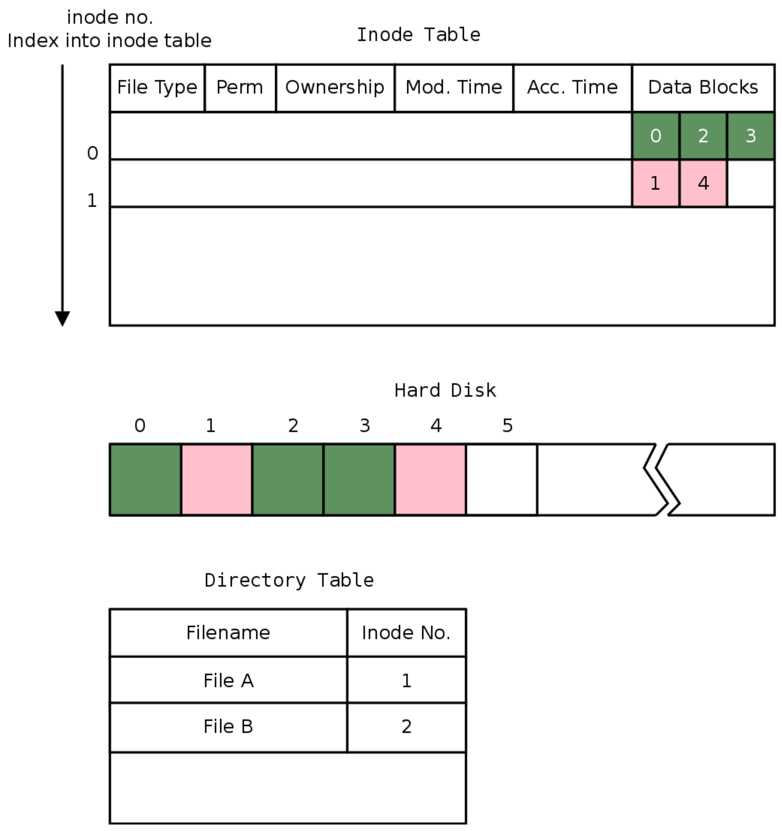




2. Unix Filesystems

- In Unix/Linux systems the table is called the inode table.
- The inode no. is the index into the inode table.
- The inode table, contains the file meta information, like the file type, permissions, ownership, access time, modifiction time, etc.
- The inode table, does not contain the file name.
- The directory files, are also like regular files, except that the contents of a directory file is a table that maps filenames to inode nos.
- There is one entry for each file present in the directory.
- When the user refers to file present in a directory, the filesystem
  - looks into the directory table

- gets the inode no., corresponding to the file
- looks into the inode table
- gets the data blocks of the file



2.1. Hard Links

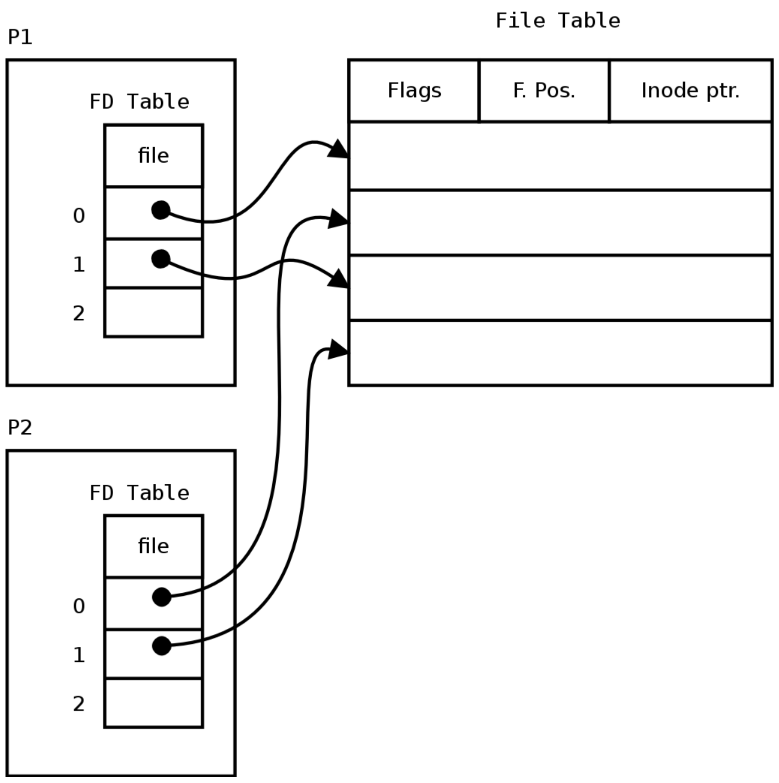
- When there are two directory table entries, that point to the same inode table entry, what we have is a link.
- The inode table has a field, that indicates the no. of links to a file.
- Each time a link is created the link count is incremented.
- Each time a link is deleted the link count is decremented. When the link count drops down to zero, the file content is freed, and the inode table entry is freed.
- Disadvantages of hard links:
  - Hard links can be created to files only within the same filesystem.
  - Hard links to directories can result in loops within the filesytem. Programs that recurse through directories like `find` and `du`, become complex if they are loops within filesystem. Hence hard links to directories are not allowed in Linux.

2.2. Symbolic Links

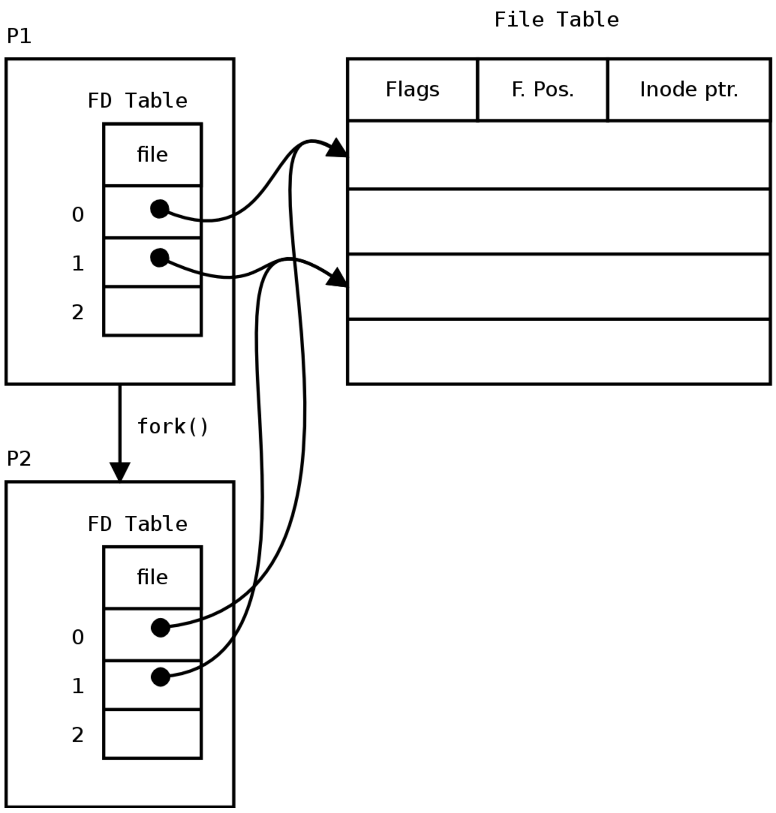
- Symbolic links are just like regular files, they have a separate entry in the inode table.
- The file type in the inode table indicates that the file is not a regular file but a symbol link.
- The field in the inode table, which is generally used for storing the blocks where the file is stored, now contains the path that the symbolic link is pointing to.
- The permissions on the symbolic link is irrelevant. The ownership of the symbolic link is used only when the parent directory has the sticky bit is set.

3. Open Files

- Every time a file opened an entry is created in a system global table, called file table.
- The file table among the other things has the following information
  - the flags specified while opening the file
  - the file position pointer
  - a pointer to the inode table entry
- Each process, has a per-process fd table. The fd table maps, file descriptors to the file table entries.



- If two processes open the same file, two different entries are created in the file table, and the file table entry is not shared.
- When a parent process, creates a child process by forking, the fd table, is copied. Now the parent fd table and the child fd table, point to the same entry in the file table.
- Writes to the file, will get interleaved.



4. File System Formats

- There are various ways of storing files and meta information about the files in the storage medium.
- The various different filesystem formats are listed below
  - EXT2, EXT3, EXT4 - defacto filesystem in Linux
  - Btrfs - designed to replace EXT filesystem in Linux, with advance filesystem features
  - NTFS - defacto filesystem in Windows
  - FAT - defacto filesystem in MS-DOS
  - ISO9660 - designed for CD-ROMs
  - UFS - descendant of the original filesystem used in Unix
  - ZFS - designed by Sun Microsystems for Solaris

5. Mounting

- Mounting is a two step process.
- Mounting associates a block device with a filesystem driver.
- The external filesystem is then attatched to the existing filesystem tree at the mount point.
- The general syntax of the mount command is given below.

```
mount -t <fs> <block-device> <mount-point>
```

- The currently mounted filesystems can be viewed by invoking the mount command without arguments.

```
mount
```

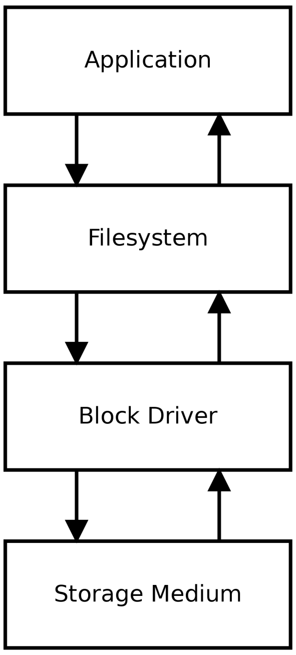
- A mounted filesystem can be unmounted using the `umount` command.

```
umount <mount-point>
```

6. Storage Stack

- When the application has to retrieve a file, it asks the file system layer.
- The file system layer looks into its inode table, and determines the sectors in which the file is present.
- The file system layer then asks the block driver to retreive the sectors.
- The block driver instructs the hardware to retreive the sectors.
- The retreived sectors are then propagated up the stack.

Figure 1. Storage Stack



6.1. Accessing Hard Disks

- The storage stack when using the hard disk with an ext2 filesystem.
- The ext2 filesystem can be created on the hard disk using `mkfs.ext2` command.

```
$ mkfs.ext2 /dev/sda1
```

- The filesystem can be mounted using the `mount` command.

```
$ mount /dev/sda1 /mnt/hd
```

Figure 2. Ext2 filesystem on Harddisk

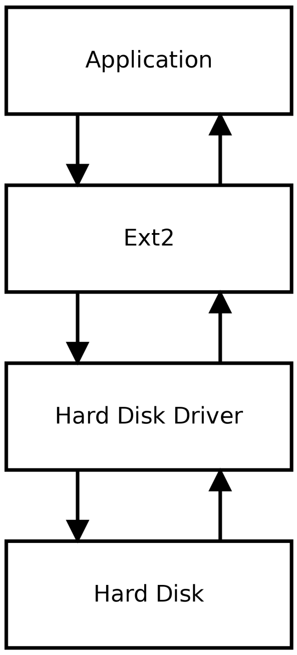
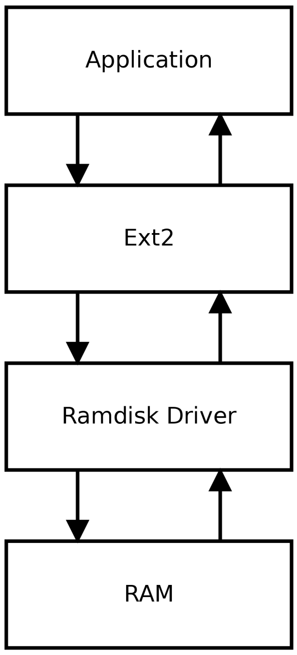


Figure 3. Ext2 filesystem on Ramdisk



6.2. Using Ramdisks

- The Ramdisk block driver treats a portion of the RAM as a storage area.
- The storage stack when using the ramdisk with an ext2 filesystem.
- The device file that corresponds to the ramdisk is `/dev/ram0`.
- The filesystem can be created and mounted as in the case of the hard disk.
- The contents of the ramdisk are lost after a system reboot.

7. Further Reading

- StackOverflow: Why hard links not allowed to directories in UNIX/Linux [http://stackoverflow.com/q/7720592]
- Unix & Linux Stack Exchange: Why hard links not allowed to directories in UNIX/Linux [http://stackoverflow.com/q/7720592]