Chapter-2 : UNIX file system

File system, Types of file, File naming convention, Parent – Child relationship, HOME variable, inode number, Absolute pathname, Relative pathname,

Significance of dot (.) and dotdot (..), Displaying pathname of the current directory (pwd), Changing the current directory (cd), Make directory (mkdir), Remove directories (rmdir),

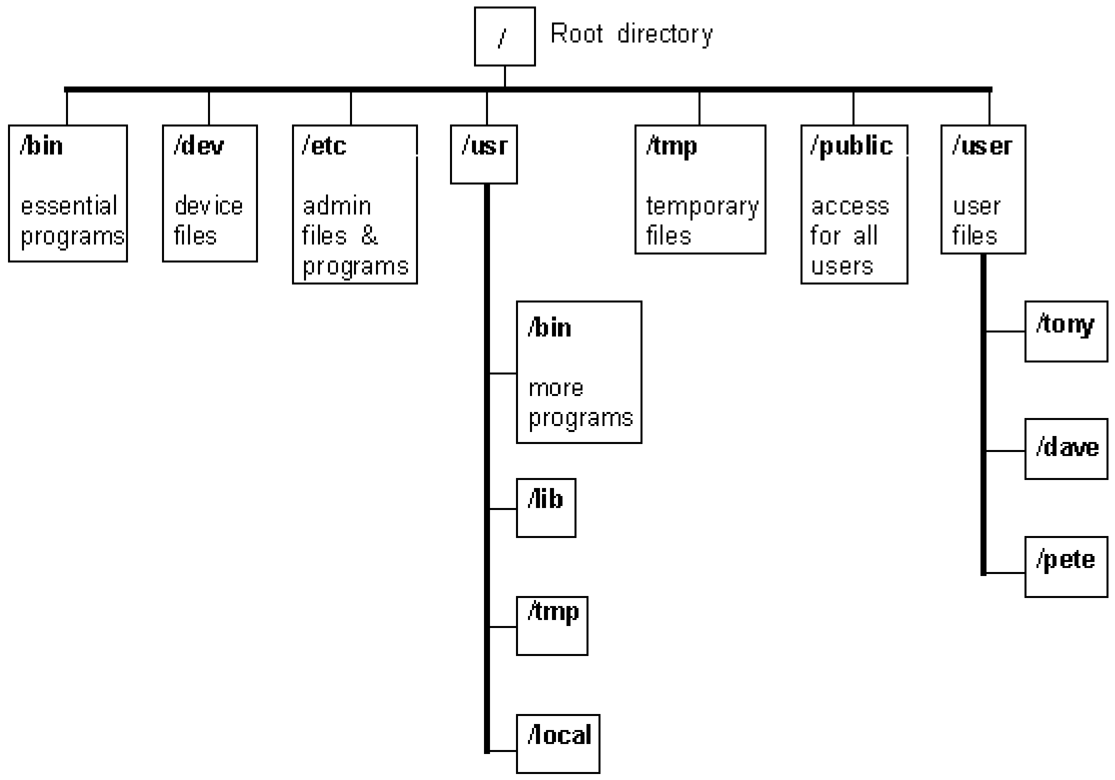
Listing contents of directory (ls),

Very brief idea about important file systems of UNIX: /bin, /usr/bin, /sbin, /usr/sbin, /etc, /dev, /lib, /usr/lib, /usr/include, /usr/share/man, /temp, /var, /home

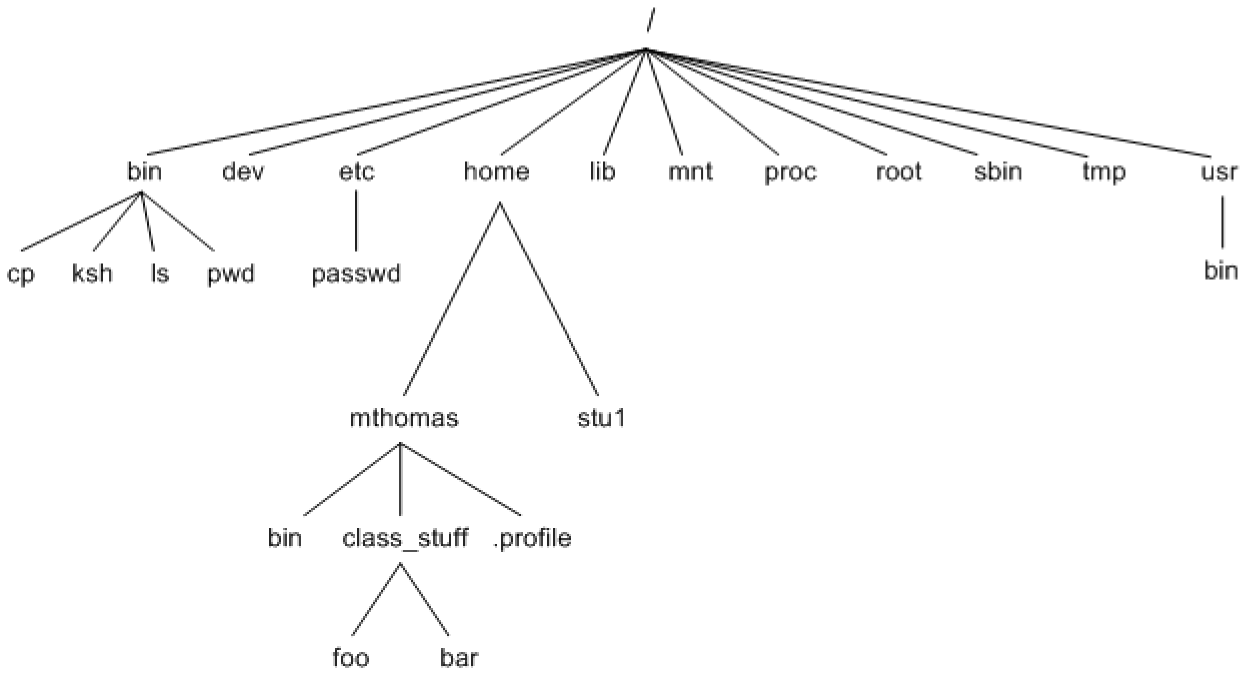
File System and parent-child relationship:

The Unix File system: -

All files in Unix are related to one another .It is a collection of all related files (ordinary, directory and device files) organized in a multi-level hierarchical structure known as a **directory** tree. The top of the file system is called **root** and it is represented by a **" / "** (front slash).



Directories and files have parent-child relationship.



<u>Directories or Files and their description –</u>

- **/:** The slash / character alone denote the root of the file system tree.
- **/bin :** Stands for "binaries" and contains certain fundamental utilities, binary executable files , where different types Unix commands are store in this directory such as **ls** or **cp**, which are generally needed by all users.

- **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices (I/O devices).

- **/etc :** it contains the configuration file of the systems and also details of users information (user name, password etc)

- **/usr:** Originally the directory holding user home directories, its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc.

- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.

- **/usr/lib:** Stores the required libraries and data files for programs stored within /usr or elsewhere.

- **/usr/include:** Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language

- **/tmp :** A place for temporary files. Many systems clear this directory upon startup.

- **/home:** Contains the home directories for the users. All created users are stored here. For example- student1, student2 etc.
- **/lib:** Contains system libraries, and some critical files such as kernel modules or device drivers. contains all library files in a binary form.

- **/var:** A short for “variable.” A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files. The var(variable) part at the file system contains print jobs and details of incoming and outgoing mail information.

- **/var/log:** Contains system log files.

- **/var/mail :** The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.

- **/var/spool :** Spool directory. Contains print jobs, mail spools and other queued tasks.

- **/var/tmp :** A place for temporary files which should be preserved between system reboots.

- **/mnt :** Stands for “mount”. Contains file system mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) file systems, CD-ROM/DVD drives, and so on.

- **/root :** The home directory for the super user “root” – that is, the system administrator.

- **/boot :** Contains all the files that are required for successful booting process.

- **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.

- **/proc :** procfs virtual file system showing information about processes as files.

- **/usr/share/man** : Manual pages (Directory man1.man2,….man8)

Directory	**Description**
man1	User programs (optional)
man2	System calls (optional)
man3	Library calls (optional)
man4	Special files (optional)
man5	File formats (optional)
man6	Games (optional)
man7	Miscellaneous (optional)
man8	System administration (optional)

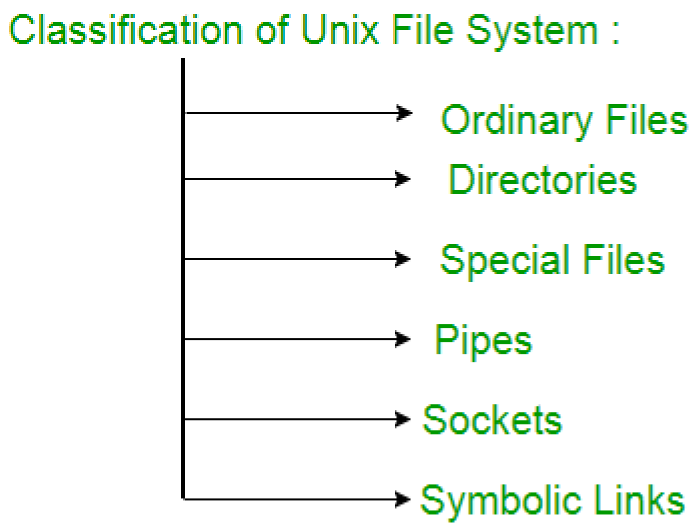
<u>HOME variable:</u>

What is Home variable in Linux?

HOME contains the path to the **home** directory of the current user. This **variable** can be used by applications to associate configuration files and such like with the user running it.

For example: /home/student1, /home/student2

Types of Unix files – The UNIX files system contains several different types of files:



1. **Ordinary (Regular) files** – An ordinary file is a file on the system that contains data (text, or program instructions). An ordinary file itself is divided into two parts **(a) Text file (**printable characters), **(b) Binary file** (printable and nonprintable characters that cover the entire ASCII range 0-255).

2. **Directory files-** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
   A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory.

   Each entry has two components.

   (i) The Filename
   (ii) A unique identification number for the file or directory (called the **inode number**).

3. **Device/Special Files –** Used to represent a real physical device such as a printer, tape drive or terminal used for Input/output (I/O) operations. They appear in a file system just like an ordinary file or a directory.

4. **Pipes –** UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another.
   A Unix pipe provides a one-way flow of data.**The output or result of the first command sequence is used as the input to the second command sequence**. To make a pipe, put a **vertical bar (|)** on the command line between two commands.
   For example: **who | wc –l**

5. **Sockets –** A UNIX socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework.

6. **Symbolic Link –** Symbolic link is used for referencing some other file of the file system. Symbolic link is also known as **Soft link**. It contains a text form of the path to the file it references **Symbolic links** are much like Windows **shortcuts**. They are like an alias that points to the real object in the file system. If we delete the source file or move it to a different location, symbolic file will not function properly.

File Naming Conventions in UNIX/Linux:

A file name, also called a filename, is a string (i.e., a sequence of characters) that is used to identify a file.

A file is a collection of related information that appears to the user as a single, contiguous block of data and that is retained in *storage*, e.g., a hard disk drive (HDD), floppy disk, optical disk or magnetic tape.

Names are given to files on Unix-like operating systems to enable users to easily identify them and to facilitate finding them again in the future.

File Name Length: UNIX allows up to 255 characters in the file name.

Allowable Characters in Filename: Only the "/" (**forward slash**) is not allowed in filenames because it is used as the pathname separator (DOS uses the back slash "\").

Characters to avoid: ? @ # $ ^ * ( ) ` [ ] \ | ; ' " < >

You can use spaces or tabs in filenames if you enclose the names in quotation marks on the command line but they are hard to work with. Use underscores or periods to get visual separation.

Example: . "this is my file" or my_file_is_this_one or here.is.another.file

Don't use - or + as the first character of a filename. Many commands use the - or + to introduce options or switches.

Filenames starting with "." are used by the system to make names **invisible (hidden)** to normal directory listings. Typically, preferences or configuration files are "hidden" using a "." prefix.

An example is ".signature" used for your electronic signature in E-mail.

Pathnames (/) : Unix uses the forward slash "/" as the pathname separator. UNIX's top directory is called the root directory and is indicated by "/".

In Unix-like operating systems, directories are just a special type of file, and thus their naming conventions are similar to those for ordinary files. The major exception is the root directory, whose name is always a forward slash (/).

However, file names are only a convenience for users, and such operating systems identify files by their **inodes,** which are numbers that are stored on the HDD in ***inode tables*** and which exist for all types of files, rather than by their names or locations in directories.

This is somewhat analogous to the ***domain names*** that are used on the Internet to identify web sites. The names are only for the convenience of human users of the system, and each site is identified by the network by a set of numbers referred to as **an *IP address***.

UNIX / Linux Inodes:

Linux Inodes:

The **inode** (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory.

An Inode number is a uniquely existing number for all the files in Linux and all Unix type systems.

When a file is created on a system, a file name and Inode number is assigned to it.

Generally, to access a file, a user uses the file name but internally file name is first mapped with respective Inode number stored in a table.

Inode Contents:

An Inode is a data structure containing metadata about the files.

Inodes contain the following information:

- **File type** - file, folder, executable program etc.
- **File size**
- **Time stamp** - creation, access, modification times
- **File permissions** - read, write, execute
- **Access control list** - permissions for special users/groups
- **File protection flags**
- **File location** - directory path where the file is stored
- **Link count** - number of hard links to the inode
- **Additional file metadata**
- **File pointers** - addresses of the storage blocks that store the file contents

NOTE: An inode does not contain the filename or the actual data. When a file is created in the UNIX file system, it is assigned an inode number and a filename. This linked pair allows the filename to be changed without affecting the file ID in the system.

Inode Number

Each Inode has a unique number and Inode number can be seen with the help of **ls -li** command.

```
$ ls –li

total 1

  844424930564508 -rwxrwxrwx 1 Debasis Goswami None  0 Aug 12 11:32 aa.txt
12947848928690251 -rw-r--r-- 1 Debasis Goswami None 16 Aug 31 16:52 file.txt
```

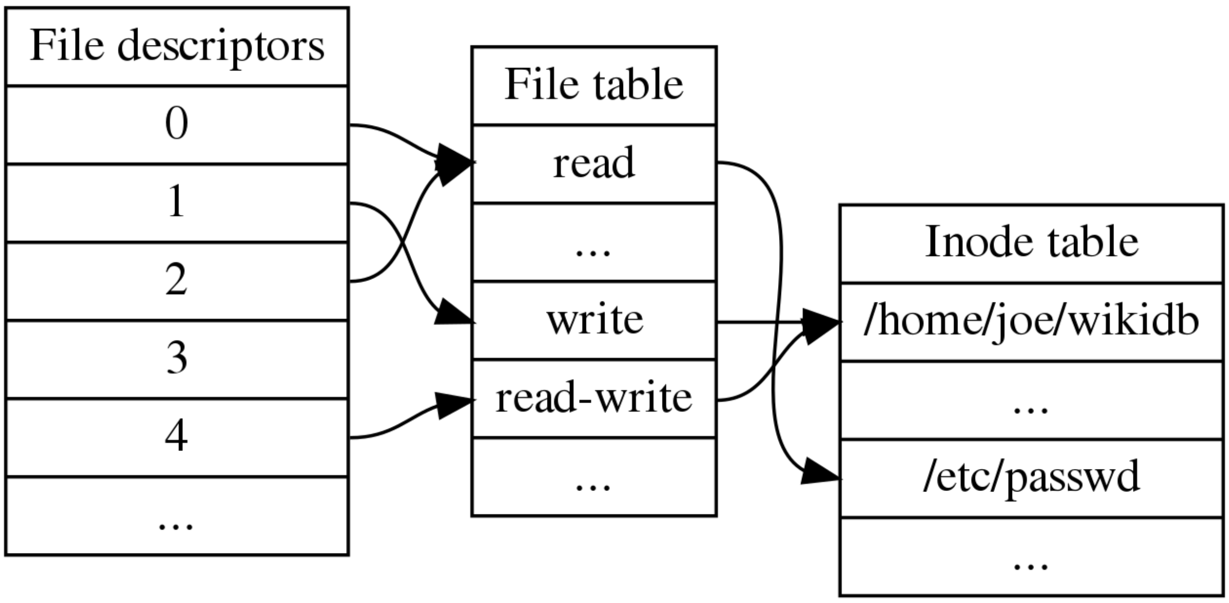
File descriptor

In Unix and related computer operating systems, a **file descriptor** (**FD**, less frequently **fildes**) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket. File descriptors form part of the POSIX application programming interface.

A file descriptor is a non-negative integer, generally represented in the C programming language as the type `int` (negative values being reserved to indicate "no value" or an error condition).

Each Unix process (except perhaps a daemon) should expect to have three standard POSIX file descriptors, corresponding to the three standard streams:

Integer value	Name	<`unistd.h`> symbolic constant[1]	<`stdio.h`> file stream[2]
0	Standard input	`STDIN_FILENO`	`stdin`
1	Standard output	`STDOUT_FILENO`	`stdout`
2	Standard error	`STDERR_FILENO`	`stderr`



File descriptors for a single process, file table and inode table. Note that multiple file descriptors can refer to the same file table entry (e.g., as a result of the dup system call[3]:104 and that multiple file table entries can in turn refer to the same inode (if it has been opened multiple times; the table is still simplified because it represents inodes by file names, even though an inode can have multiple names).
File descriptor 3 does not refer to anything in the file table, signifying that it has been closed.

Q:What is inode table?
Answer: An **inode** is an entry in **Inode table**, containing information (the metadata) about a regular file and directory. An **inode** is a data structure on a traditional Unix-style file system such as ext3 or ext4.

Q: What is ext3 and ext4 file system?
Answer: ext4 stands for fourth extended **file system**. It was introduced in 2008. ... You can also mount an existing **ext3** fs as **ext4** fs (without having to upgrade it). Several other new features are introduced in **ext4**: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc.

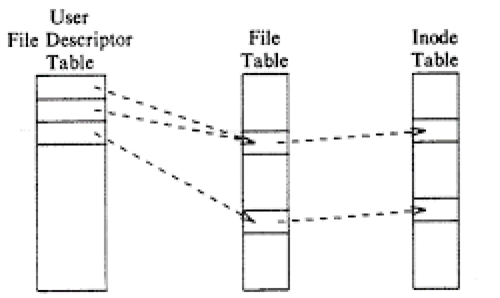
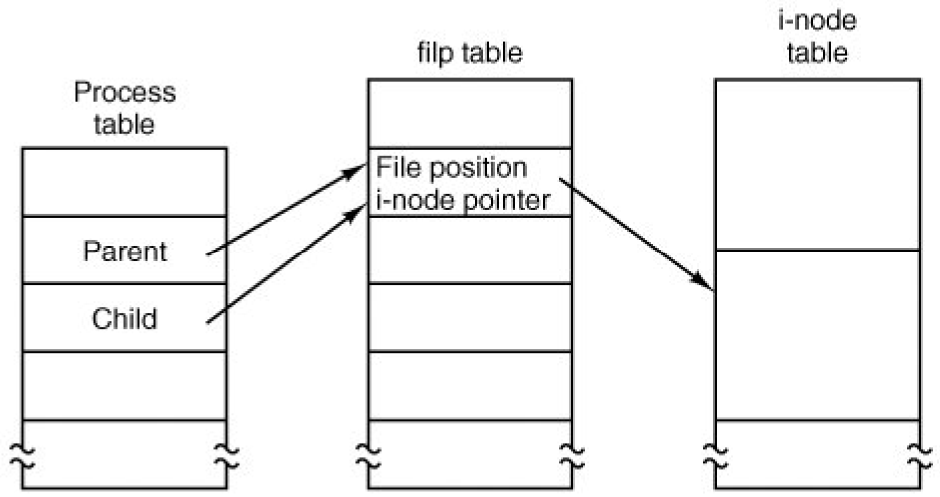
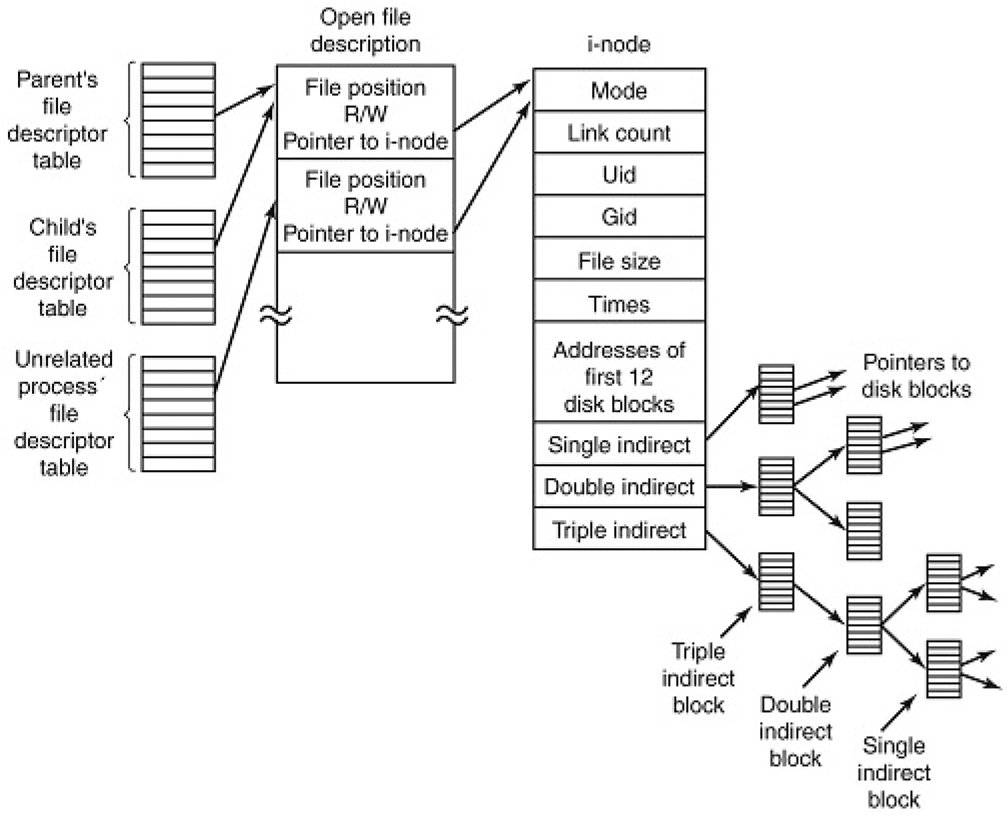
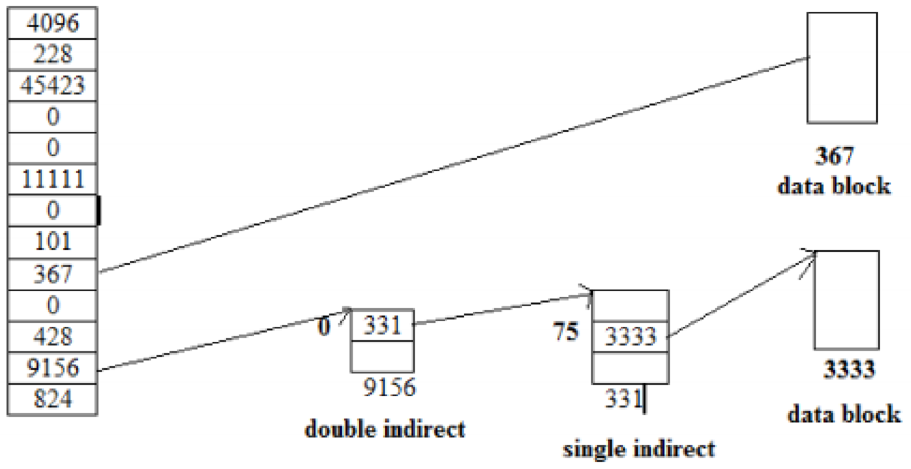


Figure 2.2. File Descriptors, File Table, and Inode Table


Process table
Parent
Child
filp table
File position
i-node pointer
i-node table


Parent's file descriptor table
Child's file descriptor table
Unrelated process´ file descriptor table
Open file description
File position
R/W
Pointer to i-node
File position
R/W
Pointer to i-node
i-node
Mode
Link count
Uid
Gid
File size
Times
Addresses of first 12 disk blocks
Single indirect
Double indirect
Triple indirect
Pointers to disk blocks
Triple indirect block
Double indirect block
Single indirect block



Block Layout of a sample File and its Inode

Several blocks entries in the inode are 0,meaning that the logical block entries contain no data. This happens if no process ever wrote data into the file at any byte offsets corresponding to those blocks and hence the block numbers remain at their initial value 0. No disk space is wasted for such blocks.

<u>Absolute and Relative Pathnames in UNIX:</u>

An **absolute path** is defined as specifying the location of a file or directory from the root directory(/). In other words, we can say that an absolute path is a complete path from start of actual file system from / directory.

<u>Relative path</u>:

Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and **never starts with a /.**

Example of Absolute and Relative Path:

Suppose you are currently located in home/BCA and you want to change your directory to home/BCA/NAME.

Changing directory with relative path concept :

1. $pwd
2. /home/BCA
3. $cd NAME

4. $pwd
5. /home/BCA/NAME

Changing directory with absolute path concept:

6. $pwd
7. /home/BCA
8. $ cd /home/BCA?NAME
9. $pwd
10. /home/BCA/NAME

Q: What is the difference between absolute path and relative path?

Answer:

An **absolute** or full **path** points to the same location **in a** file system, regardless of the current working directory. To do that, it must include the root directory.
By contrast, a **relative path** starts from some given working directory, avoiding the need to provide the full **absolute path**.

Significance of dot (.) and dotdot (..),

- **~ (tilde)** means the user's home **directory**, usually /home/username.
- **. (dot)** means the current **directory** you're in.
- **.. (dot dot)** means the parent **directory** of the current **directory** you're in

Directories with Space in Their Names :

If the directory you want to change to has spaces in its name, you should either surround the path with quotes or use the backslash (\) character to escape the space:

```
cd 'Dir name with space'
cd Dir\ name\ with\ space
```

Example:

```
$ ~
-bash: /home/Debasis Goswami: Is a directory


$ pwd
/home/Debasis Goswami

$ cd ..

$ pwd
/home
```

Example:

```
$ pwd
/home/Debasis Goswam

$ ls -R
.:
aa.txt  ab.txt  file.txt  Test

./Test:
ab.txt  KOLKATA  x.txt

./Test/KOLKATA:
```

Example:

```
$ pwd
/home/Debasis Goswami/Test

$ echo "Test file">ab.txt

$ ls
```

```
ab.txt

$ cat ab.txt
Test file

$ pwd
/home/Debasis Goswami/Test

$ cd

$ pwd
/home/Debasis Goswami


$ cp /home/Debasis\ Goswami/Test/ab.txt . (Here copy ab.txt file in current directory by
using dot (.) )

$ ls
aa.txt  ab.txt  file.txt  Test

$ cat ab.txt
Test file
```

Displaying pathname of the current directory (pwd):

pwd – Display or print name of current/working directory with the complete path starting from root (/).

SYNOPSIS

pwd [OPTION]..

Example:

```
$ pwd
/home/Debasis Goswami/Test/KOLKATA


$ type pwd
pwd is a shell builtin( Internal Command)
```

Changing the current directory (cd):

cd command in linux known as change directory command. It is used to change current working directory.

Syntax:

```
$ cd [directory]
```
To move inside a subdirectory : to move inside a subdirectory in linux we use
```
$ cd [directory_name]
```

Different functionalities of cd command :

- **cd /:** this command is used to change directory to the root directory, The root directory is the first directory in your filesystem hierarchy.
  ```
  $ cd /
  ```

cd ~ : this command is used to change directory to the home directory.
```
$ cd ~
```
or
```
$ cd
```
cd : this commad also work same as cd ~ command.

cd .. : this command is used to move to the parent directory of current directory, or the directory one level up from the current directory. “..” represents parent directory.
```
$ cd ..
```

Directories with Space in Their Names

If the directory you want to change to has spaces in its name, you should either surround the path with quotes or use the backslash (`\`) character to escape the space:

```
cd 'Dir name with space'
cd Dir\ name\ with\ space
```

cd “dir name”: This command is used to navigate to a directory with white spaces.Instead of using double quotes we can use single quotes then also this command will work.
```
$ cd "dir name"
```

You can also navigate to the same directory by using its absolute path:

```
cd /home/username/Downloads
```

In short, if the path starts with a slash (`/`) it is the absolute path to the directory.

Make directory (mkdir):

mkdir - make directories

SYNOPSIS

```
mkdir [OPTION]... DIRECTORY...
```

DESCRIPTION

Create the DIRECTORY(ies), if they do not already exist.

EXAMPLES

Example-1:

Creates a new directory called mydir whose parent is the current directory.

```
$ mkdir mydir
```

output:

```
$ls
mydir
```

Example-2:

Create the mydir directory, and set its permissions such that all users may read, write, and execute the contents.

$ mkdir -m a=rwx mydir   **OR** $mkdir –m 777 mydir

output:

```
# ls -l
total 24
-rw-r--r-- 1 ubuntu ubuntu  212 Jan  8 12:43 file.txt
drwxrwxrwx 2 root   root   4096 Jan  8 15:27 mydir
```

Example-3:

How to create multiple directories at one time:

```
$ mkdir test1 test2 test3
```

output:

```
$ls
test1  test2  test3
```

Example-4:

How to create several subdirectories at one time:

```
$ mkdir -p /home/test/test1/test2/test3/test4
```

output:

```
# ls -R /home/test/
/home/test/:
test1

/home/tutor/test1:
test2
```

```
/home/tutor/test1/test2:
test3

/home/tutor/test1/test2/test3:
test4

/home/tutor/test1/test2/test3/test4:
```

Multiple subdirectory(s) under current directories create(-p):

```
$ mkdir -p -v BCA/Batch1 BCA/Batch2
mkdir: created directory 'BCA'
mkdir: created directory 'BCA/Batch1'
mkdir: created directory 'BCA/Batch2'

$ ls -R BCA
BCA:
Batch1  Batch2

BCA/Batch1:

BCA/Batch2:


$ mkdir  -p -v  MSIT/BCA/Batch1 MSIT/BCA/Batch2
MSIT/MCA/Batch/test1/test2
mkdir: created directory 'MSIT'
mkdir: created directory 'MSIT/BCA'
mkdir: created directory 'MSIT/BCA/Batch1'
mkdir: created directory 'MSIT/BCA/Batch2'
mkdir: created directory 'MSIT/MCA'
mkdir: created directory 'MSIT/MCA/Batch'
mkdir: created directory 'MSIT/MCA/Batch/test1'
mkdir: created directory 'MSIT/MCA/Batch/test1/test2'

$ ls -R MSIT
MSIT:
BCA  MCA

MSIT/BCA:
Batch1  Batch2
```

```
MSIT/BCA/Batch1:

MSIT/BCA/Batch2:

MSIT/MCA:
Batch

MSIT/MCA/Batch:
test1

MSIT/MCA/Batch/test1:
test2

MSIT/MCA/Batch/test1/test2:
```

Remove directories (rmdir):

NAME

rmdir - remove empty directories

SYNOPSIS

```
rmdir [OPTION]... DIRECTORY...
```

DESCRIPTION

Remove the DIRECTORY(ies), if they are empty.

-p, --parents	remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'.
-v, --verbose	output a diagnostic for every directory processed.

EXAMPLES

Example-1:

rmdir command will delete the empty directories. i.e directory without any sub-directories or files:

```
$ rmdir test
```

Example-2:

To Delete Nested Empty Directories in Linux:

```
$ rmdir -p –v dir1/dir2/dir3
```

```
$ rmdir -p -v MSIT/BCA/Batch1 MSIT/BCA/Batch2
rmdir: removing directory, 'MSIT/BCA/Batch1'
rmdir: removing directory, 'MSIT/BCA'
rmdir: failed to remove directory 'MSIT/BCA': Directory not empty
rmdir: removing directory, 'MSIT/BCA/Batch2'
rmdir: removing directory, 'MSIT/BCA'
rmdir: removing directory, 'MSIT'
rmdir: failed to remove directory 'MSIT': Directory not empty
```

```
$ ls -R MSIT
MSIT:
MCA

MSIT/MCA:
Batch

MSIT/MCA/Batch:
test1

MSIT/MCA/Batch/test1:
test2

MSIT/MCA/Batch/test1/test2:
```

```
$ rmdir -pv MSIT/MCA/Batch/test1/test2
rmdir: removing directory, 'MSIT/MCA/Batch/test1/test2'
rmdir: removing directory, 'MSIT/MCA/Batch/test1'
rmdir: removing directory, 'MSIT/MCA/Batch'
rmdir: removing directory, 'MSIT/MCA'
rmdir: removing directory, 'MSIT'
```

Listing contents of directory (ls):

The **ls** is the list command in Linux. It will show the full list or content of your directory.

```
$ ls -l

total 126
-rw-r--r--  1 Debasis Goswami None    64 Sep 16 17:11 a.c
-rwxr-xr-x  1 Debasis Goswami None 40766 Sep 16 17:10 a.exe
-rw-r--r--  3 Debasis Goswami None    12 Oct 18 11:42 a.txt
-rwxr-xr-x  1 Debasis Goswami None 40766 Sep 16 17:11 aa.exe
-rw-r--r--  1 Debasis Goswami None    28 Sep 23 16:24 aa.txt
drwxr-xr-x+ 1 Debasis Goswami None     0 Oct 18 11:20 BCA
drwxr-xr-x+ 1 Debasis Goswami None     0 Oct 18 11:47 BCA2
```

```
$ ls -l
total 126
-rw-r--r--  1 Debasis Goswami None    64 Sep 16 17:11 a.c
-rwxr-xr-x  1 Debasis Goswami None 40766 Sep 16 17:10 a.exe
-rw-r--r--  1 Debasis Goswami None    12 Oct 18 11:42 a.txt
-rwxr-xr-x  1 Debasis Goswami None 40766 Sep 16 17:11 aa.exe
-rw-r--r--  1 Debasis Goswami None    28 Sep 23 16:24 aa.txt
drwxr-xr-x+ 1 Debasis Goswami None     0 Oct 18 11:20 BCA
drwxr-xr-x+ 1 Debasis Goswami None     0 Oct 18 11:47 BCA2
-rw-r--r--  1 Debasis Goswami None    61 Sep  8 12:22 dd.txt
-rw-r--r--  1 Debasis Goswami None     0 Oct 18 11:47 error_logfile
-rw-r--r--  1 Debasis Goswami None    23 Sep 30 15:51 file.txt
drwxr-xr-x+ 1 Debasis Goswami None     0 Sep  6 12:10 Test
-rw-r--r--  1 Debasis Goswami None    62 Sep 10 17:11 test.c
-rwxr-xr-x  1 Debasis Goswami None 40766 Sep 10 17:25 test.exe
```

Linux ls -l --block-size=[SIZE]

If you want to display the file size of your list in a particular format or size, then you can use this command. Just put the size in place of [SIZE] as per your requirement.

Syntax:

```
ls -l --block-size=[SIZE]
```

Example:

```
ls -l --block-size=M
```

Let's see the output below.

```
$ ls -l --block-size=k
total 126K
-rw-r--r--  1 Debasis Goswami None  1K Sep 16 17:11 a.c
-rwxr-xr-x  1 Debasis Goswami None 40K Sep 16 17:10 a.exe
-rw-r--r--  1 Debasis Goswami None  1K Oct 18 11:42 a.txt
-rwxr-xr-x  1 Debasis Goswami None 40K Sep 16 17:11 aa.exe
-rw-r--r--  1 Debasis Goswami None  1K Sep 23 16:24 aa.txt
drwxr-xr-x+ 1 Debasis Goswami None  0K Oct 18 11:20 BCA
drwxr-xr-x+ 1 Debasis Goswami None  0K Oct 18 11:47 BCA2
-rw-r--r--  1 Debasis Goswami None  1K Sep  8 12:22 dd.txt
-rw-r--r--  1 Debasis Goswami None  0K Oct 18 11:47 error_logfile
-rw-r--r--  1 Debasis Goswami None  1K Sep 30 15:51 file.txt
drwxr-xr-x+ 1 Debasis Goswami None  0K Sep  6 12:10 Test
-rw-r--r--  1 Debasis Goswami None  1K Sep 10 17:11 test.c
-rwxr-xr-x  1 Debasis Goswami None 40K Sep 10 17:25 test.exe
```

You can replace [SIZE] with the following measures:

- K = Kilobyte
- M = Megabyte
- G = Gigabyte
- T = Terabyte
- P = Petabyte
- E = Exabyte
- Z = Zettabyte
- Y = Yottabyte