Chapter 2: UNIX files

Files are the basic building blocks of any OS. When we execute any command in UNIX the kernel fetches the corresponding executable file from the file sys loads its instructions to memory and creats a process.

* Different types of files are:

Regular File : They may be text or binary files, ie, A file can be executed as it is interpreted when it is a text file and the file is converted to binary notations and interpreted when it is a binary file. These files can be read or written by the users with an appropriate access function.
Ex: rm, cat, dir, ls, lp etc.....

Directory File : This is like a file folder that contains other files including sub-directory files. This provides the user to organise the file into some hierarchical structure based on their relationship The directory files may be created using mkdir and removed using rmdir

FIFO File : It is a special pipe device file which provides a temporary buffer for two or more processes to communicate by writing data to and from the buffer. A FIFO file can be created by
$ mkfifo /usr/temp_fifo

Device File : The files that communicate with the H/W devices are called device files. They can

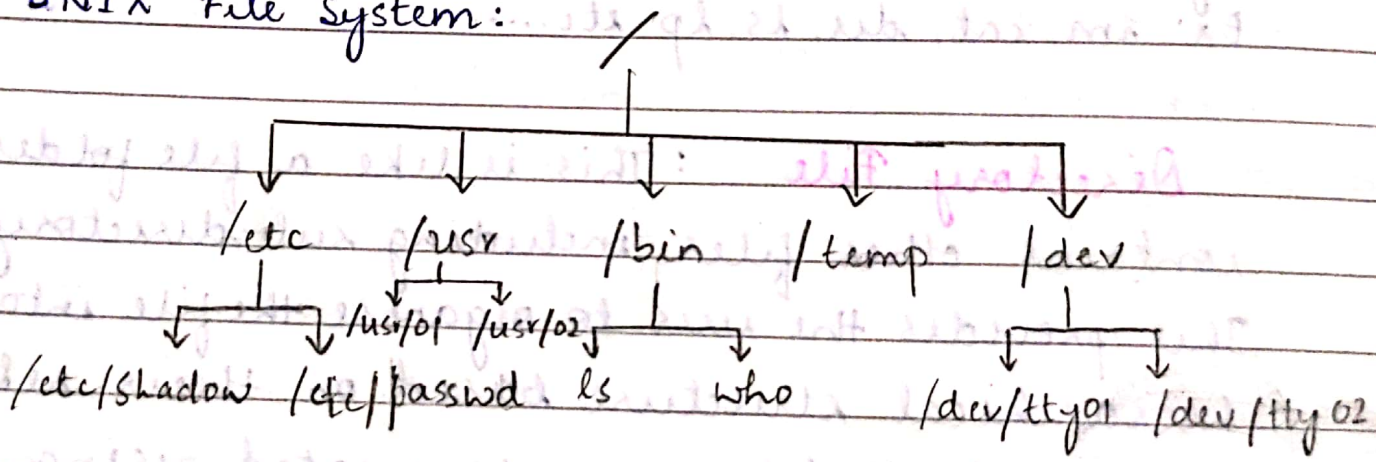
be categorized into character device file: This file represents a phy. device that transmits data in a character based manner.
Eg: Dot Matrix Printer
Block Device Files: This file represents a phy. device that transmits data, a block at a time.
Eg: Hard disk, Floppy disk.

A device file can be created as follows:

```
$mknod   /dev/chdsk   c   125   7
```

- `$mknod` → Command
- `/dev/chdsk` → file-name
- `c` → type
- `125` → addr of device driver
- `7` → device referring to

* UNIX File System:



The characteristics of UFS are:
1. A Hierarchical Structure
2. Consistent Treatment of File Data
3. Ability to create and delete a file
4. Dynamic growth of a file / or files
5. Protection of file data
6. Treatment of peripheral devices

The commonly used file systems are
/etc : It stores system admin files
/etc/passwd : It stored all user information
/etc/shadow : It stores user password.
/bin : It stores all system programs
/dev : It stores all character and block device files
/usr/include : It stores all header files.
/usr/lib : It stores all library information
/temp : It stores all temporary files created by the user.

* FILE ATTRIBUTES: (UNIX and POSIX)

File Type : It specifies the type of the file
Access Permission : It specifies access permission of user, group and others
Hard Link Count : It specifies the no. of hard links of a file
UID : It specifies file owner user ID
GID : It specifis the file group ID
File Size : It specifies the size of the file in bytes
Last Access Time : It specifies the time of the file that was accessed recently
Last Modification Time : It specifies the time of the file that was modified.
Last Changed Time : It specifies the time of changing the file access permissions for UID, GID and others
Inode No. : This specifies the system inode no. of a file.

File System ID: This specifies the file system ID stating where the file is stored

* INODE IN UNIX SYSTEMS:

The internal representation of a file is given by Inode which contains the description of the file data and other information such as file owner, access time etc.... In UNIX file system, a file has an Inode table which keeps track of all the files present in it. Each entry of Inode table is an Inode record which contains all the attributes of a file including unique inode no and the address where the data is stored.

Mapping of filenames to Inode is done through directory files in a hierarchical way where the permissions are checked at each level of hierarchy till it matches the Inode no.

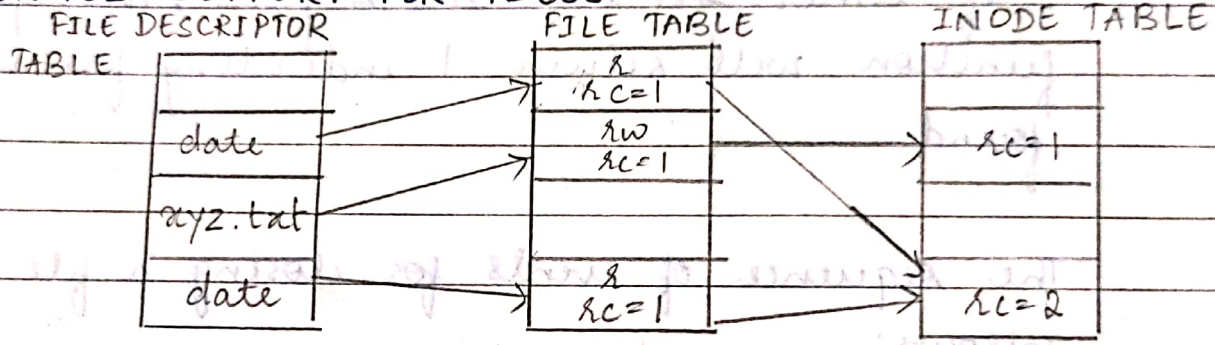
* APPLICATION PROGRAM INTERFACE TO FILES:

UNIX and POSIX systems provide an interface to the file in the following ways.

1. Files are identified by Pathnames.
2. Files must be created before they are used.
3. Files must be opened before they can be accessed by Application Program, By using the interface open().
4. read() and write() system calls are used to read the data from and write data to opened files.

5. A process may open atmost OPEN-MAX files of any type at one time.
6. File attributes can be queried by stat() or fstat() system calls.
7. The attributes of a file can be changed by chmod, chown, utime and link() system call
8. File hardlinks can be removed by unlink() system call.

* KERNEL SUPPORT FOR FILES



a file

The process for opening is as follows:

1. The kernel will search the process file descriptor table and looks for the first unused entry. If an entry is found, that entry will be designated to reference the file.
2. The Kernel will 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:

- Process File descriptor table entry will be set to point the file table entry.
- The file table entry will be set to point the Inode Table entry while the Inode is present.

• The file table entry contains file ptr to all opened files.

• The file table entry will also contain an open mode and an reference count which specifies how many file descriptors are referencing the same process.

• The reference count specifies how many file table entries are pointed to the inode in the inode table.

If either STEP 1 or STEP 2 fails, the open function will return -1 indicating file not found.

The sequence of events for closing a file are as follows:

1. The Kernel sets the corresponding file descriptor table entry as unused
2. It decrements the reference count in the corresponding file table entry by 1. If rc is still non-zero, go to STEP 6.
3. The file table entry is marked as unused
4. The reference count in the corresponding inode table is decremented by 1
5. If the count is still non-zero go to STEP 6
5. If the link count of the inode is not zero it returns the caller with the success status otherwise it marks the inode table entry as unused and deallocates all the files
6. The process continues till rc=0.

* File And Record Locking:

Unix System V and POSIX.1 use the 'fcntl' API for file and record locking. This API can be used to impose read or write locks on either a segment or entire file.

```
#include <fcntl.h>
int fcntl(int fdesc, int cmd, ...);
```

where,

fdesc : Is a file descriptor of opened file

The various command flags used for locking are:

F_SETLK : It sets a File Lock.

F_SETLKW : It sets a File Lock and blocks the calling process until the lock is acquired.

F_GETLK : It queries as to which process locked a specified region of the file

The 3rd argument is a structure "struct flock" which comes into picture

when either of the cmds are used.

This structure specifies the region of the file where the lock is to be set, unset or query.

The structure is:

```
struct flock
{
  short  l_type;
  short  l_whence;
  off_t  l_start;
  off_t  l_len;
  pid_t  pid;
};
```

- l_type : It specifies what lock is to set on files.

  Different locks are:

  F_RDLCK : It sets Read Lock on specified region

  F_WRLCK : It sets Write Lock on specified region

  F_UNLCK : It unlocks a specified region.

- l_whence: It specifies the various reference addresses. Those are:

  SEEK_CUR, SEEK_SET, SEEK_END

- l_start : It specifies the offset from l_whence location.

- l_len: It specifies how many bytes are there in the locked region.

- pid: It specifies 'pid' of a process which has locked the file

* Directory File API's

Directory file is a record oriented file where each record stores filename and Inode no. of a file that resides in the directory.

```
#include<sys/stat.h>
#include<unistd.h>

int mkdir(const char *pathname, mode_t perm);
```

where

pathname: is the name of the directory

perm: specifies the access permission of user, group, and others.

The following functions are defined in both Direct and Dirent in <sys/dir.h> & <dirent.h> respectively

```
DIR *opendir(const char *pathname);
Dirent *readdir(DIR *fdesc);
int closedir(DIR *fdesc);
void rewinddir(DIR *fdesc);
```

* Device File API's

Device Files are used to interface physical devices with Application Programs. The mknod API must be called by a process with superuser privileges when a process reads or writes to a device file, kernel uses Major and Minor device number of the file to carry out data transfer.

```
#include <sys/stat.h>
#include <unistd.h>

int mknod(const char *pathname, mode_t perm,
                                int deviceno);
```

where

pathname : specifies the pathname of the file

perm : specifies access permissions for U, G & O

devicenno : specifies major and minor API.

* FIFO file API's

They are called as named files, where output of one process is passed to the input of another process. This process takes place only between related processes.

```
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int mkfifo(const char *pathname, mode_t perm);
```

where:

pathname : Specifies the pathname of the file

perm : Specifies the access permissions for U, G and O

* Symbolic Link File API's

A symbolic link is an Indirect Pointer to a file, unlink the hard links which points directly to the Inode of the files.

```
#include<sys/stat.h>
#include<unistd.h>
int symlink(const char *o_link, const char *n_link);
```

where

o_link : Is the original link

n_link : Specifies the new link to be created.