UNIT-V

Files and Directories in UNIX, File Structure, File System Implementation of Operating System Functions, File permission, Basic Operation on Files, Changing Permission Modes, Standard files, Processes Inspecting Files, Operating On Files.

Files and Directories in UNIX:

A directory is a file the solo job of which is to store the file names and the related information. All the files, whether ordinary, special, or directory, are contained in directories.

Unix uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character (/), and all other directories are contained below it.

Home Directory

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command −

```
$cd ~
$
```

Here **~** indicates the home directory. Suppose you have to go in any other user's home directory, use the following command −

```
$cd ~username
$
```

To go in your last directory, you can use the following command −

```
$cd -
$
```

Absolute/Relative Pathnames

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.

Following are some examples of absolute filenames.

```
/etc/passwd
/users/sjones/chem/notes
/dev/rdsk/Os3
```

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this −

```
chem/notes
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory −

```
$pwd
/user0/home/amrood

$
```

Listing Directories

To list the files in a directory, you can use the following syntax −

```
$ls dirname
```

Following is the example to list all the files contained in **/usr/local** directory −

```
$ls /usr/local

X11       bin          gimp       jikes       sbin
ace       doc          include    lib         share
atalk     etc          info       man         ami
```

Creating Directories

We will now understand how to create directories. Directories are created by the following command −

```
$mkdir dirname
```

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command −

```
$mkdir mydir
$
```

Creates the directory **mydir** in the current directory. Here is another example −

```
$mkdir /tmp/test-dir
$
```

This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, −

```
$mkdir docs pub
$
```

Creates the directories docs and pub under the current directory.

Creating Parent Directories

We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, **mkdir** issues an error message as follows −

```
$mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
$
```

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example −

```
$mkdir -p /tmp/amrood/test
$
```

The above command creates all the required parent directories.

Removing Directories

Directories can be deleted using the **rmdir** command as follows −

```
$rmdir dirname
$
```

Note − To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can remove multiple directories at a time as follows −

```
$rmdir dirname1 dirname2 dirname3
$
```

The above command removes the directories dirname1, dirname2, and dirname3, if they are empty. The **rmdir** command produces no output if it is successful.

Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below −

```
$cd dirname
$
```

Here, **dirname** is the name of the directory that you want to change to. For example, the command −

```
$cd /usr/local/bin
$
```

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path −

```
$cd ../../home/amrood
$
```

Renaming Directories

The **mv (move)** command can also be used to rename a directory. The syntax is as follows −

```
$mv olddir newdir
$
```

You can rename a directory **mydir** to **yourdir** as follows −

```
$mv mydir yourdir
$
```

The directories . (dot) and .. (dot dot)

The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory, often referred to as the parent directory.

If we enter the command to show a listing of the current working directories/files and use the **-a option** to list all the files and the **-l option** to provide the long listing, we will receive the following result.

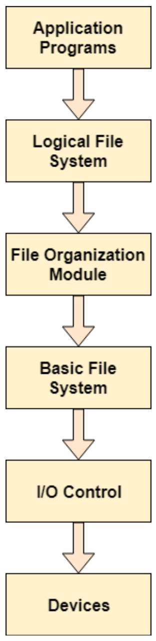
```
$ls -la
drwxrwxr-x    4    teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x    60   root              1536  Jul 13 14:18 ..
----------    1    teacher   class   4210  May 1 08:27 .profile
-rwxr-xr-x    1    teacher   class   1948  May 12 13:42 memo
$
```

File Structure:

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order

to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.

- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

FILE SYSTEM IMPLEMENTATION OF OPERATING SYSTEM FUNCTIONS:

Numerous on-disk and in-memory configurations and structures are being used for implementing a file system. These structures differ based on the operating system and the file system but applying some general principles. Here they are portrayed below:

- A boot control block usually contains the information required by the system for booting an operating system from that volume. When the disks do not contain any operating system, this block can be treated as empty. This is typically the first chunk of a volume. In UFS, this is termed as the boot block; in NTFS, it is the partition boot sector.
- A volume control block holds volume or the partition details, such as the number of blocks in the partition, size of the blocks or chunks, free-block count along with free-block pointers. In UFS, it is termed as superblock; in NTFS, it is stored in the master file table.
- A directory structure per file system is required for organizing the files. In UFS, it held the file names and associated 'inode' numbers. In NTFS, it gets stored in the master file table.
- The FCB contains many details regarding any file which includes file permissions, ownership; the size of file and location of data blocks. In UFS, it is called the inode. In NTFS, this information gets stored within the master file table that uses a relational database (RDBM) structure, using a row per file.

Advantages :

1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

Disadvantages:

If we access many files at same time then it results in low performance.

We can **implement** file system by using two types data structures :

1. **On-disk Structures –**

   Generally, they contain information about total number of disk blocks, free disk blocks, location of them and etc. Below given are different on-disk structures:

   1. **Boot Control Block –**

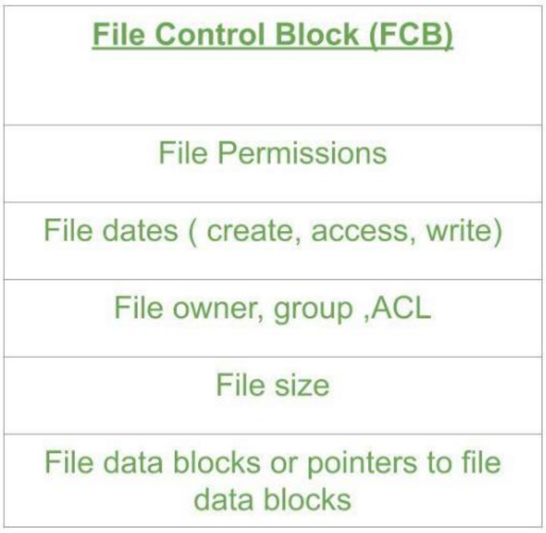
      It is usually the first block of volume and it contains information needed to boot an operating system. In UNIX it is called boot block and in NTFS it is called as partition boot sector.

2. **Volume Control Block –**

   It has information about a particular partition ex:- free block count, block size and block pointers etc.In UNIX it is called super block and in NTFS it is stored in master file table.
3. **Directory Structure –**

   They store file names and associated anode numbers. In UNIX, includes file names and associated file names and in NTFS, it is stored in master file table.
4. **Per-File FCB –**

   It contains details about files and it has a unique identifier number to allow association with directory entry. In NTFS it is stored in master file table.



File Control Block (FCB)
File Permissions
File dates (create, access, write)
File owner, group ,ACL
File size
File data blocks or pointers to file data blocks

2. In-Memory Structure:

They are maintained in main-memory and these are helpful for file system management for caching. Several in-memory structures given below :

1. **Mount Table –**
   It contains information about each mounted volume.
2. **Directory-Structure cache –**
   This cache holds the directory information of recently accessed directories.
3. **System wide open-file table –**
   It contains the copy of FCB of each open file.
4. **Per-process open-file table –**
   It contains information opened by that particular process and it maps with appropriate system wide open-file.

Directory Implementation:

5. **Linear List –**

   It maintains a linear list of filenames with pointers to the data blocks. It is time-consuming also. To create a new file, we must first search the directory to be sure that no existing file has the same name then we add a file at end of the directory. To delete a file, we search the directory for the named file and release the space.

To reuse the directory entry either we can mark the entry as unused or we can attach it to a list of free directories.

6. **Hash Table –**

   The hash table takes a value computed from the file name and returns a pointer to the file. It decreases the directory search time. The insertion and deletion process of files is easy. The major difficulty is hash tables are its generally fixed size and hash tables are dependent on hash function on that size.

Basic Operations on Files:

A file is a collection of logically related data that is recorded on the secondary storage in the form of sequence of operations. The content of the files are defined by its creator who is creating the file. The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

Types of File Operations :

1.Create operation:

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open operation:

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write operation:

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read operation:

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek operation:

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. Delete operation:

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate operation:

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close operation:

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. Append operation:

This operation adds data to the end of the file.

10. Rename operation:

This operation is used to rename the existing file.

Changing Permission Modes :

The chmod command enables you to change the permissions on a file. You must be superuser or the owner of a file or directory to change its permissions.

You can use the chmod command to set permissions in either of two modes:

- **Absolute Mode –** Use numbers to represent file permissions. When you change permissions by using the absolute mode, you represent permissions for each triplet by an octal mode number. Absolute mode is the method most commonly used to set permissions.
- **Symbolic Mode –** Use combinations of letters and symbols to add permissions or remove permissions.

The following table lists the octal values for setting file permissions in absolute mode. You use these numbers in sets of three to set permissions for owner, group, and other, in that order. For example, the value 644 sets read and write permissions for owner, and read-only permissions for group and other.

File Permissions in Absolute Mode

Octal Value	File Permissions Set	Permissions Description
0	---	No permissions
1	--x	Execute permission only

Octal Value	File Permissions Set	Permissions Description
2	-w-	Write permission only
3	-wx	Write and execute permissions
4	r--	Read permission only
5	r-x	Read and execute permissions
6	rw-	Read and write permissions
7	rwx	Read, write, and execute permissions

The following table lists the symbols for setting file permissions in symbolic mode. Symbols can specify whose permissions are to be set or changed, the operation to be performed, and the permissions that are being assigned or changed.

File Permissions in Symbolic Mode

Symbol	Function	Description
u	*who*	User (owner)
g	*who*	Group
o	*who*	Others
a	*who*	All
=	*operator*	Assign
+	*operator*	Add
-	*operator*	Remove
r	*permissions*	Read
w	*permissions*	Write
x	*permissions*	Execute
l	*permissions*	Mandatory locking, setgid bit is on, group execution bit is off
s	*permissions*	setuid or setgid bit is on
t	*permissions*	Sticky bit is on, execution bit for others is on

The *who operator permissions* designations in the function column specify the symbols that change the permissions on the file or directory.

who

Specifies whose permissions are to be changed.

operator

Specifies the operation to be performed.

permissions

Specifies what permissions are to be changed.

You can set special permissions on a file in absolute mode or symbolic mode. However, you must use symbolic mode to set or remove setuid permissions on a directory. In absolute mode, you set special permissions by adding a new octal value to the left of the permission triplet. The following table lists the octal values for setting special permissions on a file.

Special File Permissions in Absolute Mode

Octal Value	Special File Permissions
1	Sticky bit
2	setgid
4	setuid