Process

A process in a UNIX abstraction that enables us to look at files and programs in another way. A file is treated as a simple file when it lies in a dormant state on disk. It can also be understood as a process when it is executed. Like living organisms processes are born, they give birth to other processes and also die. Processes make things happen in UNIX.

A process is simply an instance of a running program. A process is said to be born when the program starts execution and remains alive as long as the program is active. After execution is complete, the process is said to die. A process also has a name, usually the name of the program being executed. For example, when you execute the grep command, a process named grep is created. A process can't be considered synonymous with a program. When two users run the same program, there is one program on disk but two processes in memory.

Kernel is responsible for the management of these processes. It determines the time and priorities that are allocated to processes so that multiple processes are able to share CPU resources. It provides a mechanism by which a process is able to execute for a finite period of time and then relinquish control to another process. The kernel has to sometimes store pages of these processes in the swap area of the disk before calling them again for running.

Attributes of processes are maintained by the kernel in memory in a separate structure called the process table. Two important attributes of a process are:

- The Process-id (PID) – Each process is uniquely identified by a unique integer called the PID that is alloted by the kernel when the process is born. We need this PID to control a process.
- The Parent PID (PPID) – The PID of the parent is also available as a process attribute. When several processes have the same PPID, it often makes sence to kill the parent rather than all its children separately.

Mechanism of Process Creation –

There are three distinct phases in the creation of a process and uses three important system calls – fork, exec and wait.

- fork – A process in UNIX is created with the fork system call, which creates a copy of the process that invokes it. The process image is partically identical to that of the calling process, except for a few parameters like the PID. When a process is forked in a new PID The forking mechanism is

- Exec – The child then overwrites the image that it has just created with the copy of the program that has to be executed. This is done with a system call belonging to the exec family (which has six functions), and the parent is said to exec this process. No additional process is created here; the existing program is simply replaced with the new program. This process has the same PID as the child that was just forked.
- Wait – The parent then executes the wait system call to wait for the child process to complete. It picks up the exit status of the child and then continues with its other functions. A parent may not decide to wait for the death of its child.

Running jobs in background – (&, nohup)

A multitasking system lets a user do more than one job at a time. Since there can be only one job in the foreground, the rest of the jobs have to run in background. There are two ways of doing this, – with the shell's & operator and the nohup command. nohup permits you to logout while your jobs are running but & does not allow that (except in the C and Bash shell)

Inode – Inode is a table which contains all the details which we need to know about a file – except its name and contents. Inode is accessed by inode numb
Inode contains the following attributes of a file:

- File Type (regular, directory, device etc)
- File permissions (the nine permissions and three more)
- Number of links (the number of aliases the file has)
- The UID of the owner
- The GID of the group owner
- File size in bytes
- Date and time of last modification
- Date and time of last access
- Date and time of last change of the mode
- An array of pointers that keep track of all disk blocks used

hard Links –

hen a link is copied both the original and copy occupy eparate space on disk. UNIX allows a file to have more han one name and yet maintain a single copy on disk. he file is then said to have more than one link; i.e. it as more than one name. All the names provided to a single file have one thing in common – they all have the same node number.

```
$ ls -li backup.sh restore.sh
478274 -rwxr-xr--   2 kumar metal   163 Jul 13 21:36 backup.sh
478274 -rwxr-xr--   2 kumar metal   163 Jul 13 21:36 restore.sh
```

Both "files" indeed have the same inode number, so there is actuall only one file with a single copy on disk. We can not really refer to them as two "files" but only as two "filenames". This file simply has two aliases; changes made in one alias (link) are automatically available in the others. There are two entries for this file in the directory both having the same inode numb

Creating hard links –

```
$ ln emp.lst employee
```

Here employee must not exist.

oft Links -

Hard links have two limitations:

- You can not have two linked filenames in two file systems
- You can not link a directory even within the same file system

A symbolic link does not contain the file's contents, but simply provides the pathname of the file that actually has the contents.

```
$ ln -s note note.ln
$ ln -li note note.ln
9948 -rw-r--r--   1 kumar group   ... Feb 16 14:52 note
9952 lrwxrwxrwx   1 kumar group   ... Feb 16 15:07 note.ln -> note
```

System Call -

C was created by by Dennis Ritchie to rewrite the UNIX operating system in that language. To access the services available in the operating system all UNIX system offer around 200 different functions called system calls. A system call is a routine built into the kernel and performs a very basic function that requires communication with the CPU, memory and devices. All activities related to file handeling, process and memory management and maintenance of user and system information are handled by the kernel using these system calls.

File Descriptor Table -

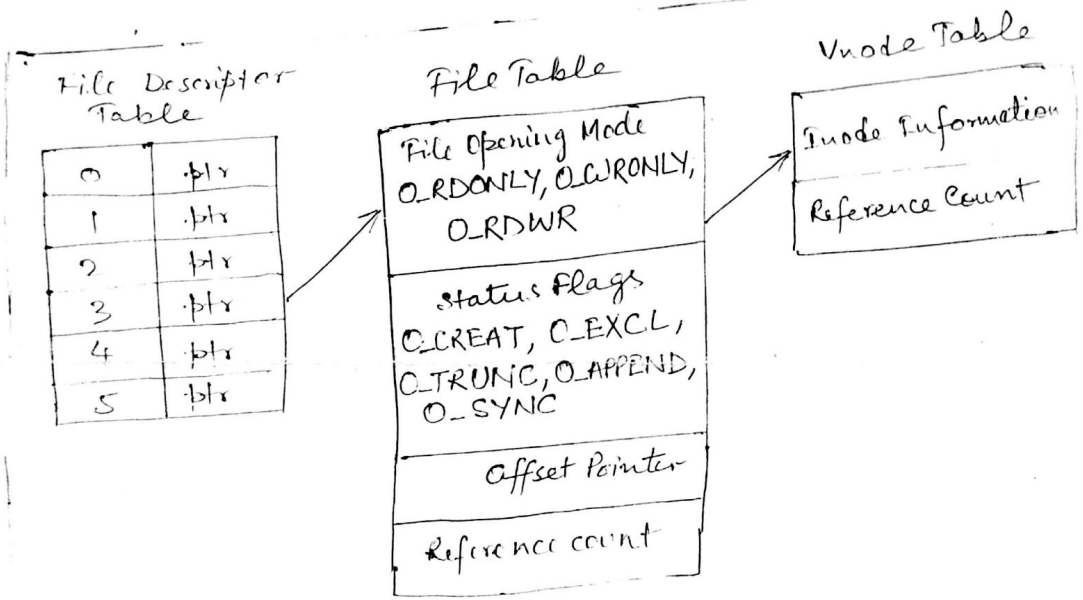
The kernel returns a non-negative integer (the file descriptor) to the process that opens the file. This file descriptor is stored along with a flag in the file descriptor table. This table is maintained separately for every process that runs in the system as a component of a separate structure called the u area. The u area contains all attributes of the currently running process.

The shell's three standard files occupy the first three slots (0, 1 and 2) in the file descriptor table. So you close desc. riptor number 1, the kernel will allocate this number to the next file that is opened. We can use this behavioral property to implement redirection. The flag FD_CLOEXEC is not used by ~~close~~ open, but by a powerful system call named fcntl to determine whether the descriptor will be closed when the process does an exec to run a separate program. By default a descriptor is not closed when the process does an exec.

File Table -

Every entry in the file descriptor table points to a file table. This table contains all data that are relevant to an opened file. It contains -

- The mode of opening (like O_RDONLY)
- The status flag (like O_CREAT, O_TRUNC etc)
- The offset pointer location that determines the byte positions to be used by the next read or write operation.
- A reference count that indicates the number of process or calls that point to this table.



Vnode Table –

The file table contains a pointer to the vnode table. Vnode abstraction was created to make it possible to access the inode in a file-system (as there may be multiple file system) independent way.
The vnode table contains all information present in the inode except that this structure is maintained in the memory. There is only a single copy of the vnode in memory. The inode on disk is updated from the information in the vnode by the kernel. If the

processes open the same file, or the same program opens the same file twice, there would be seperate entries in the file descriptor tables and two seperate file table structures, but both structures would point to the same inode table.

The inode table maintains a reference count field that signifies the number of file table entries that point to this table. When a file is deleted the kernel has to first check the reference count to see whether any process still has the file open. If the reference count is at least one, the kernel can't delete the file and release the inode though it will delete the directory entry for the file.

Init -

After a machine is powered on the system looks for all peripherals and then goes through a series of steps that may take up to a few minutes to complete the boot cycle. The exact sequence is system dependent, but the first major event is the loading of the kernel into memory. The kernel then spawns init (PID 1) which spawns further processes. Some of these processes monitor all the terminal lines, activate the network and printers. Eventually, init becomes the parent of all shells and system daemons.

A UNIX system boots to a specific state or mode and this state is called the run level. The default run level as well as the action to take for each run level are controlled by init. Each run level is normally a single digit (0 to 6) or an s or S. A distinct set of processes (mostly daemons) is scheduled to run in each of these states. Normally the system would be in any of these run levels:

0 - System shutdown
1 - System administration mode (local filesystems mounted)
2 - Multiuser mode (NFS not mounted)
3 - Full multiuser mode
5 - The graphical environment mode in Linux
6 - Shutdown and reboot mode
s or S - Single-user mode (file systems mounted)

/etc/passwd and /etc/shadow

All user information except the password encription is stored in /etc/passwd. The encrepted password is stored in /etc/shad

There are seven fields in /etc/passwd file.

- Username - The name of the user to log on to a UNIX system.
- Password - No longer stores the password encription but contains x.
- UID - The user's numerical identification.
- GID - The user's numerical group identification.
- Comment or GCOS - User details eg. name, address etc
- Home directory - The directory where the user ends up on logging in. The variable HOME is set by the login program by reading this field.
- Login shell - The first program executed after logging in. logi also sets the variable SHELL by reading this ent and also fork-execs the shell process.

for every ~~entry~~ line in /etc/passwd there is a corresponding en in /etc/shadow. The relevant line in this file could look l this:

user1:PRIhjDhiRM2Lg:12032::::::

The password encription is shown in the second field. It is impos to generate the password from this encryption.

BIOS - Turn off pnp for ISA slots

Mount Points - The directory that holds the contents of a partation is called its mount point.

Mounting & Unmounting File Systems —

U must mount a file system before you can access the files it contains. To mount a file system issue a command such as:

mount -t ext2 /dev/hdb1 /mnt/data

will mount the ext2 file system residing on the partation /dev/hdb1 as the directory /mnt/data

read only —

mount -t ext2 /dev/hdb1 /mnt/data -o ro

unmount /dev/hdb1
or
unmount /mnt/data

mount /mnt/cdrom

Printing files.

lpr file

lpr -P lj5 file ← on specific printer

Back up and Restore

tar cvf tarfile.tar file1 file2 dir1

restore

tar xvf tarfile.tar

init

2 - multiuser nonet
3- " neter
5. x-based log in

linuxconf