**The UNIX Process:**

A process in UNIX is a program in execution with definite life-time and well-defined hierarchy.

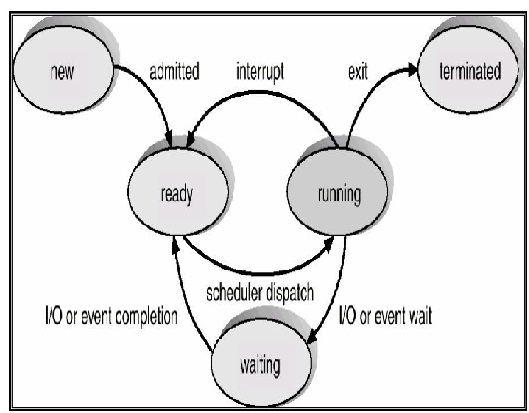
* The context of a process is a snapshot of its current run-time environment that consists of the current state of the processor registers and
* User program image - Process execution on user data structure that represents an application and is manipulated by its own functions (user mode execution).
* System image - Process execution on system’s data structures that represent the resources (memory, files, devices) and supported by the kernel routines. Depending upon resource requirement and availability, process’s states are controlled by executing the kernel routines accessed through system calls (system mode execution).
* The kernel maintains a process table to manage all processes. The two data structures per process are the user structure and the process structure.
* The kernel process is the first (root) process that comes into existence when the system is booted. Its process\_id and group\_id are both 0.
* In establishing a multi-programming environment, the kernel process  creates the init process with its process\_id of 1 and group\_id, 0, showing that process 1 is part of process 0. The init process creates and manages terminal processes for active terminals in the system.
* At the time of creation, each terminal process belongs to process group 0 and is made to execute a program called getty. Once each terminal process (now called a getty process) establishes communication with the associated terminal, it displays a login message and waits for a user to input a user name (Process group of the init process).
* When a user starts a dialogue, the getty process receives the user name and leaves the task of validating the user’s password to another program called login. The same terminal process is now called a login process.
* The login process, after validating the user’s password, calls a command line program, the login shell to run in the same process. The same terminal process is now called a shell process.
* Each shell process now establishes a new process group and becomes ready to process the user commands. A shell process is the initiating process by which each terminal maintains the user session.
* While interpreting a command, the shell creates an execution thread and then assigns the requested command program to this new process.
* Both the shell and the new process proceed independently in separate execution threads. The parent shell process normally waits until child process completes its execution.

A process require certain resources to accomplish its intended task, they are  
>> CPU time  
>> System memory  
>> Disk Files  
>> I/O devices.

**Process control blocks (PCB)**  
The OS must know all the information about a specific process in order to manage and control it. The OS maintains a table (an array of structures), called the process table, with one entry per process. These entries are called process control blocks (PCB) - also known as task control block.This entry contains information about the process' state, its program counter, stack pointer,memory allocation, the status of its open files, its accounting and scheduling information, and everything else about the process that must be saved when the process is switched from ready to running or blocked state so that it can be restarted later with the status it left off.

**Process state:**- The state may be new, ready, running, waiting, halted, and so on.  
**Program counter:**- The counter indicates the address of the next instruction to be  
executed for this process.  
**CPU registers:**- The registers vary in number and type, depending on the computer  
architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.  
**CPU-scheduling information**:- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.  
**Memory-management information**:- This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the OS.  
**Accounting information:**- This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.  
**I/O status information**:- This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

**Process state:**  
A process may present in one of the following state.  
New:- The process is being created or just created.  
Ready:- The process is waiting to be assigned/allocated to a processor time.  
Running:- Instructions of the process are being executed by processor.  
Waiting:- The process is waiting for some other event/process to execute.  
Terminated:- The process has finished its execution.



**ps command:**  
The ps (i.e., process status) command is used to provide information about the currently running processes in the system. When ps is used without any option four columns of information labeled PID,TTY, TIME and CMD for at least two processes, the shell and ps will be visible.  
PID :- The process are identified by a 5 digit number known as PID (Process Identification Number).  
TTY :- Is the name of the console or terminal that the user logged into (Stands for terminal type now  
but originally stood for teletype).  
TIME :- Is the amount of CPU time in minutes and seconds that the process has been running.  
CMD :- is the name of the command that launched the process.

The information that ps -aux provides about each process is :  
>> The user of the process,  
>> PID of the process,  
>> Percentage of CPU used by the process,  
>> Percentage of memory used by the process,  
>> VSZ (virtual size in kilobytes),  
>> RSS (real memory size or resident set size in 1024 byte units),  
>> STAT (the process state code, explained later),  
>> Starting time of the process,  
>> Length of time the process has been active  
>> The command that initiated the process.

**Types of process**  
**Parent and Child Process:**  
A process can initiate a sub process, which is a called a child process, the initiating process is referred to as its parent.The child processes, in turn create other child processes forming a tree of processes  
( which can be displayed using ps command with –forest option)  
**Orphan Process:**  
When a child process is killed, parent process gets the notification via a signal. Parent then, can continue other task. However if the parent process is killed before, its child, is called an orphan process.  
**Zombie Process:**  
When a process finished its execution and exit status not received by the parent ( or parent did not read the exit status till now), the process state becomes zombie.  
The process is dead (not to be scheduled for further execution) but cannot becompletely removed from process table, until it has been determined that exit status is no longer needed.

**Daemon Process**  
Some programs are not designed to be run with continuous user input and disconnect from the terminal when task completed. For example, a web server responds to web requests, rather than user input. Mail servers are another example of this type of application. These types of programs are known as daemons.

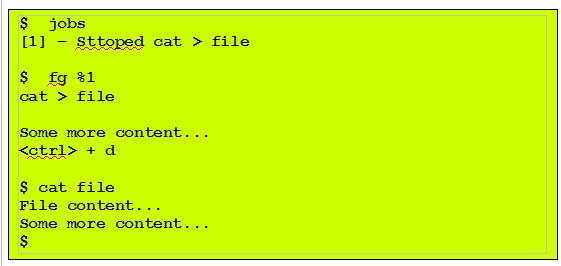
**Starting a Process:**  
Process can be started in two ways:  
**In Foreground:** By default every process starts in foreground, ie. Gets the input from keyboard and sends the output in monitor. But in this case till the process completes its execution no other process can be started in foreground.  
**In Background:** To take the advantage multiprocessing environment, a process can be started in background, so that other process can be started in the foreground without waiting for the previous process to complete execution.  
A process can be started in background by adding ampersand(&) after it.

[390119 @ INGNRILPORCL] $ $ ls | wc -l > file1 &

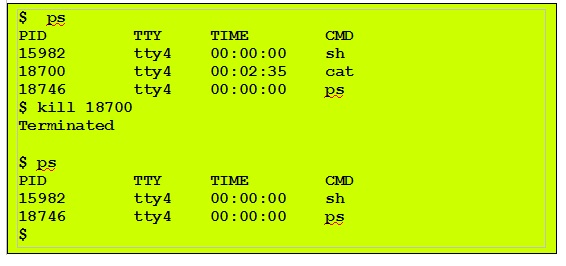
**Switching process from foreground to background:**  
A process running in foreground can be send to background using the following steps:  
>> Press + Z to suspend the job  
>> bg command puts the job in background  
>> nohup unattaches the job from the terminal\*

$ cat > file  
File content...  
+ z  
[1] + Stopped  
$ bg  
[1] + cat > file &

**Switching process from background to foreground**  
Process running in the background can be taken into foreground using the following steps:  
>> Find the job id of the process by the command jobs  
>> Use fg %to get the job to foreground



**Stopping/Killing a Process**  
A process dies(terminates ) automatically when it completes the job it intended to.  
A process can be killed abnormally with the command kill.  
>> Use the ps command to find out the process-id for the process  
>> Use the command kill to terminate it  
>> Use the command kill -9 to kill a process forcefully.



**Option settings for background process**  
While a process is executing/running , if the owner tries to log off the process ,process will get killed. Sometimes a job or command takes a long time to complete and it is required the job to be completed without interruption. This situation can be handled in two different ways:  
>> Does not allow the use to log off  
>> Continue execution in background even after the user logged off

This can be achieved using command nohup

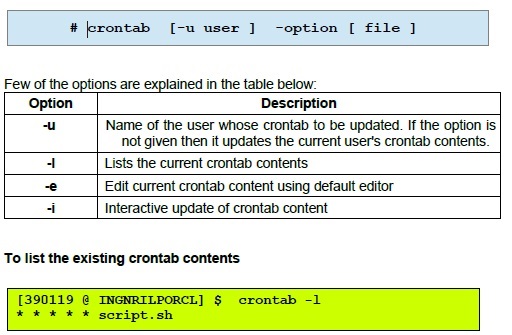
**PERIODICAL EXECUTION OF JOBS**

Sometimes it may be required to execute certain Job or task on specific time. It may be not possible for any user to start or execute the job physically on that particular time due to other business requirements. Again if similar task need to be executed periodically on a regular interval, then the task becomes very hectic.  Unix internally has solutions to handle  this type of situations using utilities such as cron and at.

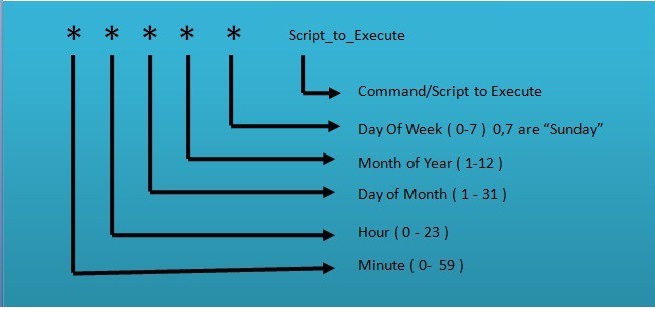
**Crontab**  
The word 'crontab' in UNIX stand for chron(chronograph) table, or time table for Unix system .  
The crontab is a special table where it is possible to specify commands and time or interval to execute the command once or repeatedly. Commands can be any executable programs, for example, a script can be written to take regular backup of all the required tables in database.  A crontab instruction can be created to execute the script on the specified time. The cron daemon reads the table and executes the commands at the times specified.

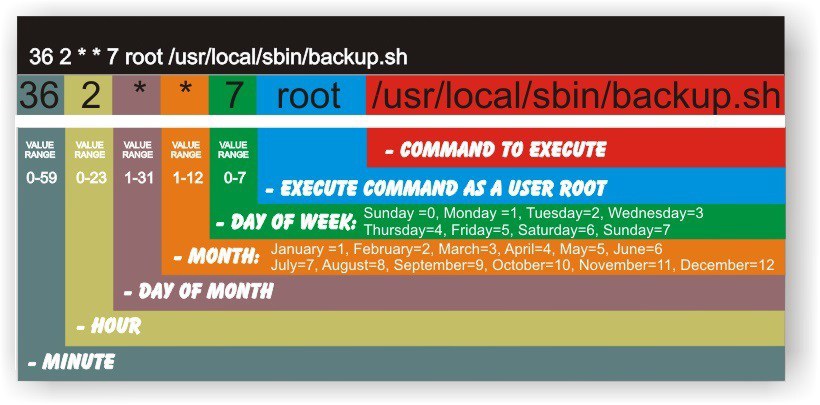
**Crontab Command Syntax**

Syntax



**Components of Crontab**  
The basic format of a crontab schedule consists of 6 fields, placed on a single line and separated by spaces, formatted as follows:  
The various fields and their utility are explained in the image below:





Example:

To executes an incremental backup shell script "takeBackup.sh" available in your home directory at 11:00  on every day the cron entry will be as below:

00 11 \* \* \* ~/takeBackup.sh

                      To  schedule  the same job for every minute, the cron entry will be as follows:

                  \* \* \* \* \* ~/takeBackup.sh