Experiment 16

Aim: Process oriented commands: ps, pstree, kill, killall (with all their options),

A process refers to a program in execution; it's a running instance of a program. It is made up of the program instruction, data read from files, other programs or input from a system user.

Types of Processes
There are fundamentally two types of processes in Linux:
Foreground processes (also referred to as interactive processes) – these are initialized and controlled through a terminal session. In other words, there has to be a user connected to the system to start such processes; they haven't started automatically as part of the system functions/services.

- To bring a background process to the foreground, enter:

```
fg
```

- If you have more than one job suspended in the background, enter:

```
fg %#
```

Replace # with the job number, as shown in the first column of the output of the jobs command.
Background processes (also referred to as non-interactive/automatic processes) – are processes not connected to a terminal; they don't expect any user input.
Adding & along with the command starts it as a background process

```
$ pwd &
```

Kill background jobs

Similar to moving jobs out of the background, you can use the same form to kill the processes by using their Job ID.

```
kill %1
```

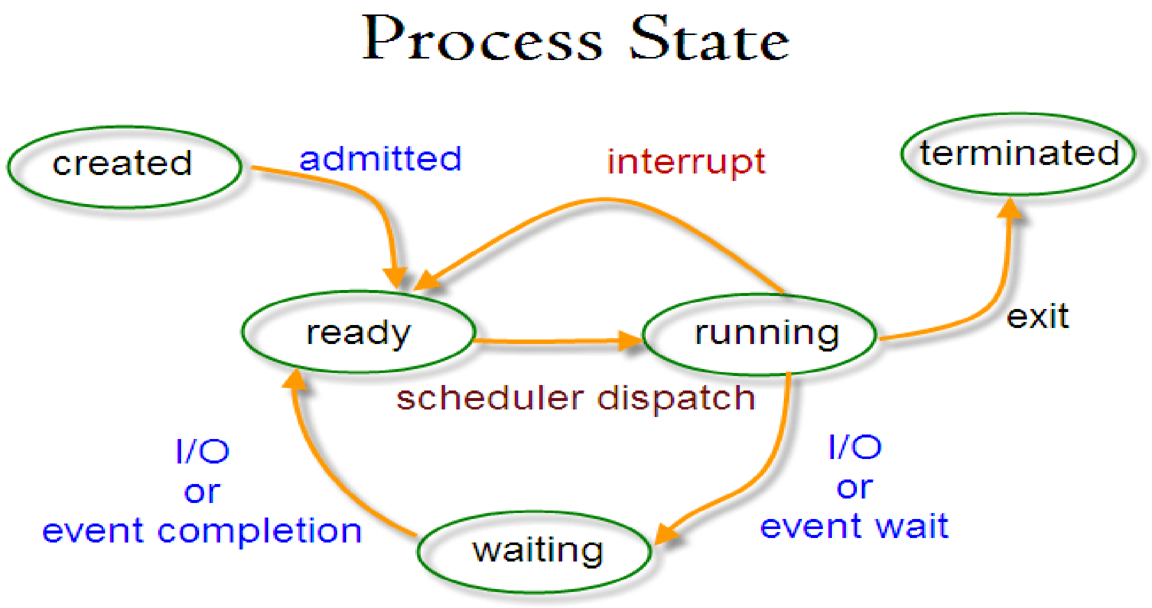
Zombie and Orphan Processes
Normally, when a child process is killed, the parent process is updated via a **SIGCHLD** signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," the **init** process, becomes the new PPID (parent process ID). In some cases, these processes are called orphan processes.

When a process is killed, a **ps** listing may still show the process with a **Z** state. This is a zombie or defunct process. The process is dead and not being used. These processes are different from the orphan processes. They have completed execution but still find an entry in the process table.

What is Daemons?

These are special types of background processes that start at system startup and keep running forever as a service; they don't die. They are started as system tasks (run as services), spontaneously. However, they can be controlled by a user via the init process.

Linux Process State



States of a Process in Linux

During execution, a process changes from one state to another depending on its environment/circumstances. In Linux, a process has the following possible states:

- Running – here it's either running (it is the current process in the system) or it's ready to run (it's waiting to be assigned to one of the CPUs).
- Waiting – in this state, a process is waiting for an event to occur or for a system resource. Additionally, the kernel also differentiates between two types of waiting processes; interruptible waiting processes – can be interrupted by signals and uninterruptible waiting processes – are waiting directly on hardware conditions and cannot be interrupted by any event/signal.
- Stopped – in this state, a process has been stopped, usually by receiving a signal. For instance, a process that is being debugged.

- Zombie – here, a process is dead, it has been halted but it's still has an entry in the process table.

Creation of Processes in Linux

A new process is normally created when an existing process makes an exact copy of itself in memory. The child process will have the same environment as its parent, but only the process ID number is different.

Using fork() and exec() Function we can create processes in the system

How Does Linux Identify Processes?

Because Linux is a multi-user system, meaning different users can be running various programs on the system, each running instance of a program must be identified uniquely by the kernel. And a program is identified by its process ID (PID) as well as it's parent processes ID (PPID), therefore processes can further be categorized into:

Parent processes – these are processes that create other processes during run-time.

Child processes – these processes are created by other processes during run-time.

The Init Process

Init process is the mother (parent) of all processes on the system, it's the first program that is executed when the Linux system boots up; it manages all other processes on the system. It is started by the kernel itself, so in principle it does not have a parent process.

The init process always has process ID of 1. It functions as an adoptive parent for all orphaned processes.

ps (Processes)

Processes are the programs that are running on your machine. They are managed by the kernel and each process has an ID associated with it called the **process ID (PID).** This PID is assigned in the order that processes are created.

Go ahead and run the ps command to see a list of running processes:

```
$ ps
PID      TTY    STAT   TIME        CMD
41230    pts/4   Ss      00:00:00    bash
51224    pts/4   R+      00:00:00    ps
```

This shows you a quick snapshot of the current processes:

- PID: Process ID
- TTY: Controlling terminal associated with the process (we'll go in detail about this later)
- STAT: Process status code
- TIME: Total CPU usage time
- CMD: Name of executable/command

If you look at the man page for ps you'll see that there are lots of command options you can pass, they will vary depending on what options you want to use - BSD, GNU or Unix. In my opinion the BSD style is more popular to use, so we're gonna go with that. If you are curious the difference between the styles is the amount of dashes you use and the flags.

```
$ ps aux
```

- The **a** displays all processes running, including the ones being ran by other users.
- The **u** shows more details about the processes.
- the **x** lists all processes that don't have a TTY associated with it, these programs will show a ? in the TTY field, they are most common in daemon processes that launch as part of the system startup.

You'll notice you're seeing a lot more fields now, no need to memorize them all, in a later course on advanced processes, we'll go over some of these again:

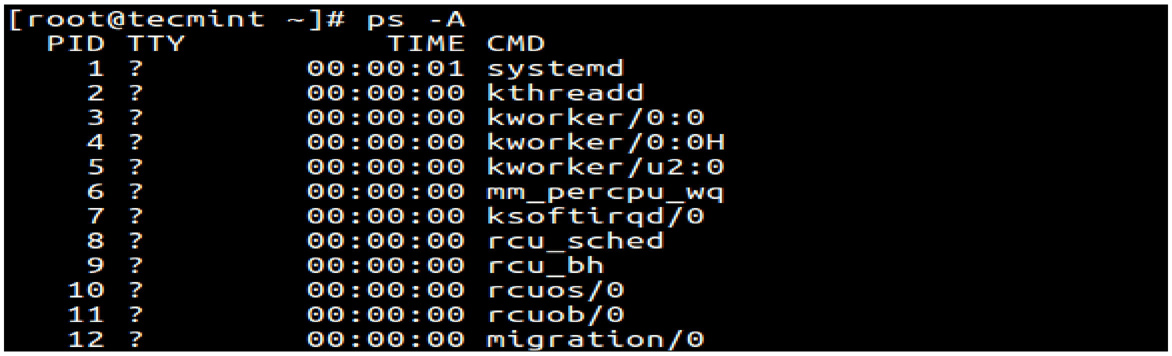
- USER: The effective user (the one whose access we are using)
- PID: Process ID
- %CPU: CPU time used divided by the time the process has been running
- %MEM: Ratio of the process's resident set size to the physical memory on the machine
- VSZ: Virtual memory usage of the entire process
- RSS: Resident set size, the non-swapped physical memory that a task has used
- TTY: Controlling terminal associated with the process
- STAT: Process status code
- START: Start time of the process
- TIME: Total CPU usage time

- COMMAND: Name of executable/command

The ps command can get a little messy to look at, for now the fields we will look at the most are PID, STAT and COMMAND.

Display every active process on a Linux system in generic (Unix/Linux) format.

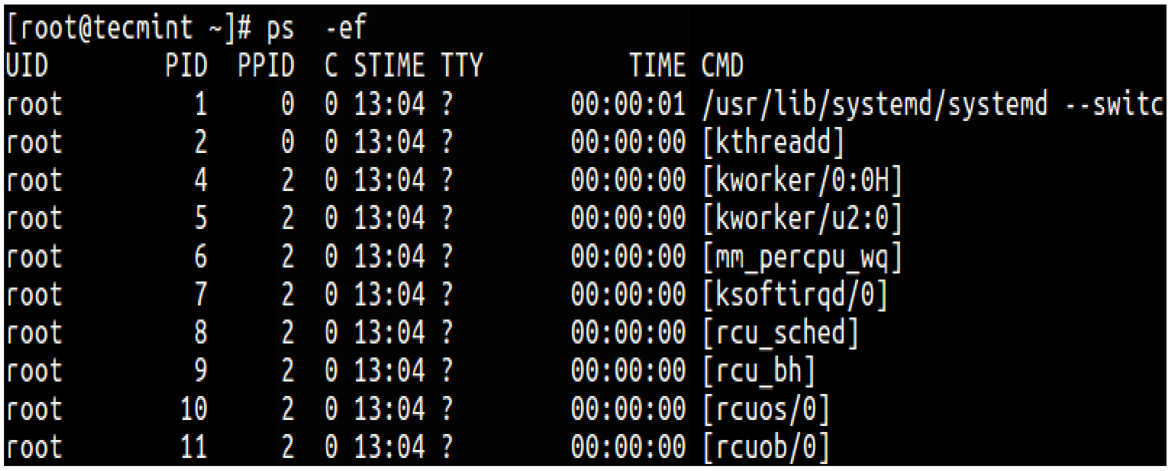
```
$ ps -A
OR
$ ps -e
```



```
[root@tecmint ~]# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 kworker/0:0
    4 ?        00:00:00 kworker/0:0H
    5 ?        00:00:00 kworker/u2:0
    6 ?        00:00:00 mm_percpu_wq
    7 ?        00:00:00 ksoftirqd/0
    8 ?        00:00:00 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 rcuos/0
   11 ?        00:00:00 rcuob/0
   12 ?        00:00:00 migration/0
```

To perform a full-format listing, add the -f or -F flag.

```
$ ps -ef
OR
$ ps -eF
```



```
[root@tecmint ~]# ps  -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13:04 ?        00:00:01 /usr/lib/systemd/systemd --switc
root         2     0  0 13:04 ?        00:00:00 [kthreadd]
root         4     2  0 13:04 ?        00:00:00 [kworker/0:0H]
root         5     2  0 13:04 ?        00:00:00 [kworker/u2:0]
root         6     2  0 13:04 ?        00:00:00 [mm_percpu_wq]
root         7     2  0 13:04 ?        00:00:00 [ksoftirqd/0]
root         8     2  0 13:04 ?        00:00:00 [rcu_sched]
root         9     2  0 13:04 ?        00:00:00 [rcu_bh]
root        10     2  0 13:04 ?        00:00:00 [rcuos/0]
root        11     2  0 13:04 ?        00:00:00 [rcuob/0]
```

Print All Processes Running as Root (Real and Effecitve ID)

The command below enables you to view every process running with root user privileges (real & effective ID) in user format.

```
$ ps -U root -u root
```

Display Processes by PID and PPID

You can list processes by PID as follows.

```
$ ps -fp 1178
```

To select process by PPID, type.

```
$ ps -f --ppid 1154
```

Make selection using PID list.

```
$ ps -fp 2226,1154,1146
```

How to use pstree command?

Basic usage is simple: all you have to do is to execute 'pstree' sans any option.

```
pstree
```

or

```
$ ps -e --forest
```

Display command line arguments along-with tree

```
$ pstree -a
```

Display PIDs

```
$ pstree -p
```

Numeric Sort using -n option

To sort processes with the same ancestor by PID instead of by name i.e. numeric sort, pass the –n options as follows:

```
$ pstree -np
```

kill (Terminate)

You can send signals that terminate processes, such a command is aptly named the kill command.

```
$ kill 12445
```

The 12445 is the PID of the process you want to kill. By default it sends a TERM signal. The SIGTERM signal is sent to a process to request its termination by allowing it to cleanly release its resources and saving its state.

How to send a custom signal?

As already mentioned in the introduction section, TERM is the default signal that kill sends to the application/process in question. However, if you want, you can send any other signal that kill supports using the **-s** command line option.

```
kill -s [signal] [pid]
```

What are the other ways in which signal can be sent?

Signals can be specified in three ways:

- **By number (e.g. -5)**
- **With SIG prefix (e.g. -SIGkill)**
- **Without SIG prefix (e.g. -kill)**

In one of the previous examples, we told you if you want to send the KILL signal, you can do it in the following way:

```
kill -s KILL [pid]
```

However, there are a couple of other alternatives as well:

```
kill -s SIGKILL [pid]
```

You can also specify a signal with the kill command:

```
$ kill -9 12445
```

This will run the SIGKILL signal and kill the process.

How to view list of all signals

```
$ kill -l
```

Differences between SIGHUP, SIGINT, SIGTERM, SIGKILL, SIGSTOP?

These signals all sound reasonably similar, but they do have their differences.

- SIGHUP - Hangup, sent to a process when the controlling terminal is closed. For example, if you closed a terminal window that had a process running in it, you would get a SIGHUP signal. So basically you've been hung up on
- SIGINT - Is an interrupt signal, so you can use Ctrl-C and the system will try to gracefully kill the process
- SIGTERM - Kill the process, but allow it to do some cleanup first
- SIGKILL - Kill the process, kill it with fire, doesn't do any cleanup
- SIGSTOP - Stop/suspend a process

How to kill all running process in one go?

In case a user wants to kill all processes that they can (this depends on their privilege level), then instead of specifying a large number of process IDs, they can simply pass the -1 option to kill.

For example: *kill -s KILL -1*

Linux killall command

The killall command lets you kill processes by name.

Q1. How to use killall command?

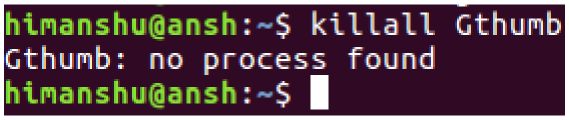
The tool's basic usage is very easy - all you have to do is to pass the name of the process as argument to killall. For example, to kill the *gthumb* process that was running on my system, I used killall in the following way:

```
killall gthumb
```

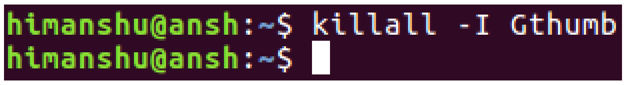
Note that in case you aren't aware of the exact name of the process, you can use the *ps* command to fetch this information.

Q2. Is killall case-sensitive?

Yes, it is. It throws an error 'no process found' error if the user fails to write the process name in correct case. Following is an example:


```
himanshu@ansh:~$ killall Gthumb
Gthumb: no process found
himanshu@ansh:~$
```

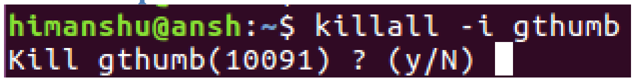
However, if you want, you can force killall to ignore case using the -I command line option.


```
himanshu@ansh:~$ killall -I Gthumb
himanshu@ansh:~$
```

Q3. How to make killall ask before terminating process?

Suppose you want the killall command to ask for user permissions before it kills a process, then you can use the -i command-line option. This will make the killall operation interactive.

For example:


```
himanshu@ansh:~$ killall -i gthumb
Kill gthumb(10091) ? (y/N)
```

Q4. How to kill all processes owned by a user?

If the requirement is to kill all processes that a specific user owns, then you can use the -u option provided by killall. Needless to say, the option requires you to specify the username for the user as its input.

```
killall -u [user-name]
```

For example:

```
killall -u himanshu
```

Q5. To kill the last background job, the following command is used:

```
kill $!
```