Process management

Aprocess 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.
- Background processes(also referred to as non-interactive/automatic processes) – are processes not connected to a terminal; they don't expect any user input.

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.

Creation of a 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.
There are two conventional ways used for creating a new process in Linux:

- Using The System() Function– this method is relatively simple, however, it's inefficient and has significantly certain security risks.
- Using fork() and exec() Function – this technique is a little advanced but offers greater flexibility, speed, together with security.

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 machine 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.

You can use the pidof command to find the ID of a process:

```
# pidof systemd

 pidof top

# pidof httpd
```

To find the process ID and parent process ID of the current shell, run:

```
$ echo $$



$ echo $PPID
```

Linux Background Jobs

To start a process in the background (non-interactive), use the `&` symbol, here, the process doesn't read input from a user until it's moved to the foreground.

```
# cloudcmd &

# jobs
```

You can also send a process to the background by suspending it using [ctrl+z] this will send the SIGSTOP signal to the process, thus stopping its operations; it becomes idle:

```
# tar -cf backup.tar /backups/*   #press Ctrl+Z
```

To continue running the above-suspended command in the background, use the bg command:

```
# bg
```

To send a background process to the foreground, use the fg command together with the job ID like so:

```
# jobs

# fg %1
```

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.

How to View Active Processes in Linux

There are several Linux tools for viewing/listing running processes on the system, the two traditional and well known are ps andtop commands:

1. ps Command

It displays information about a selection of the active processes on the system as shown below:

```
# ps

# ps -e | head
```

```
[root@tecmint ~]# ps
  PID TTY          TIME CMD
 2109 pts/0    00:00:00 bash
 2200 pts/0    00:00:01 node
 2321 pts/0    00:00:00 ps
[root@tecmint ~]# ps -e | head
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 kworker/u2:0
    7 ?        00:00:00 migration/0
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:00 rcuob/0
   10 ?        00:00:00 rcu_sched
[root@tecmint ~]#
```

List Linux Active Processes

2. top – System Monitoring Tool

top is a powerful tool that offers you a dynamic real-time view of a running system as shown in the screenshot below:

```
# top
```

```
top - 08:53:06 up 16 min,  1 user,  load average: 0.20, 0.28, 0.35
Tasks: 238 total,   1 running, 237 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.4 us,  1.0 sy,  0.0 ni, 93.0 id,  0.6 wa,  0.0 hi,  0.0 si,
KiB Mem :  3742792 total,  1144416 free,  1236544 used,  1361832 buff/ca
KiB Swap:  5631996 total,  5631996 free,        0 used.  2249948 avail M

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMM
 2469 aaronki+  20   0 1757760 168424  56580 S  11.6  4.5   0:37.61 cinn
 3691 aaronki+  20   0  479484  35208  26268 S   6.3  0.9   0:00.42 gnom
 1946 root      20   0  463000  96932  85920 S   4.3  2.6   0:21.98 Xorg
  170 root      20   0       0      0      0 S   1.3  0.0   0:00.69 kwor
    6 root      20   0       0      0      0 S   1.0  0.0   0:00.85 kwor
  921 root      20   0  449740  19428  14048 S   0.7  0.5   0:01.05 Netw
 1743 shinken   20   0 1557824  31040   6504 S   0.7  0.8   0:06.95 shin
 1817 shinken   20   0 1631460  32172   7856 S   0.7  0.9   0:04.32 shin
    7 root      20   0       0      0      0 S   0.3  0.0   0:01.17 rcu_
 1865 shinken   20   0 1632116  32604   7284 S   0.3  0.9   0:05.92 shin
 1908 shinken   20   0 1557024  30232   6556 S   0.3  0.8   0:03.24 shin
 1953 root      20   0 1633896  34552   5712 S   0.3  0.9   0:04.19 shin
 2082 shinken   20   0 1631232  29112   4728 S   0.3  0.8   0:00.20 shin
 3684 aaronki+  20   0   41908   3808   3104 R   0.3  0.1   0:00.04 top
    1 root      20   0  119696   5924   4040 S   0.0  0.2   0:01.45 syst
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthr
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.01 ksof
```

How to Control Processes in Linux

Linux also has some commands for controlling processes such as kill, pkill, pgrep and killall, below are a few basic examples of how to use them:

```
$ pgrep -u tecmint top



$ kill 2308

$ pgrep -u giri top

$ pgrep -u giri glances

$ pkill glances
```

```
$ pgrep -u giri glances
```

Sending Signals To Processes

The fundamental way of controlling processes in Linux is by sending signals to them. There are multiple signals that you can send to a process, to view all the signals run:

```
$ kill -l
```

To send a signal to a process, use the kill, pkill or pgrep commands we mentioned earlier on. But programs can only respond to signals if they are programmed to recognize those signals.

And most signals are for internal use by the system, or for programmers when they write code. The following are signals which are useful to a system user:

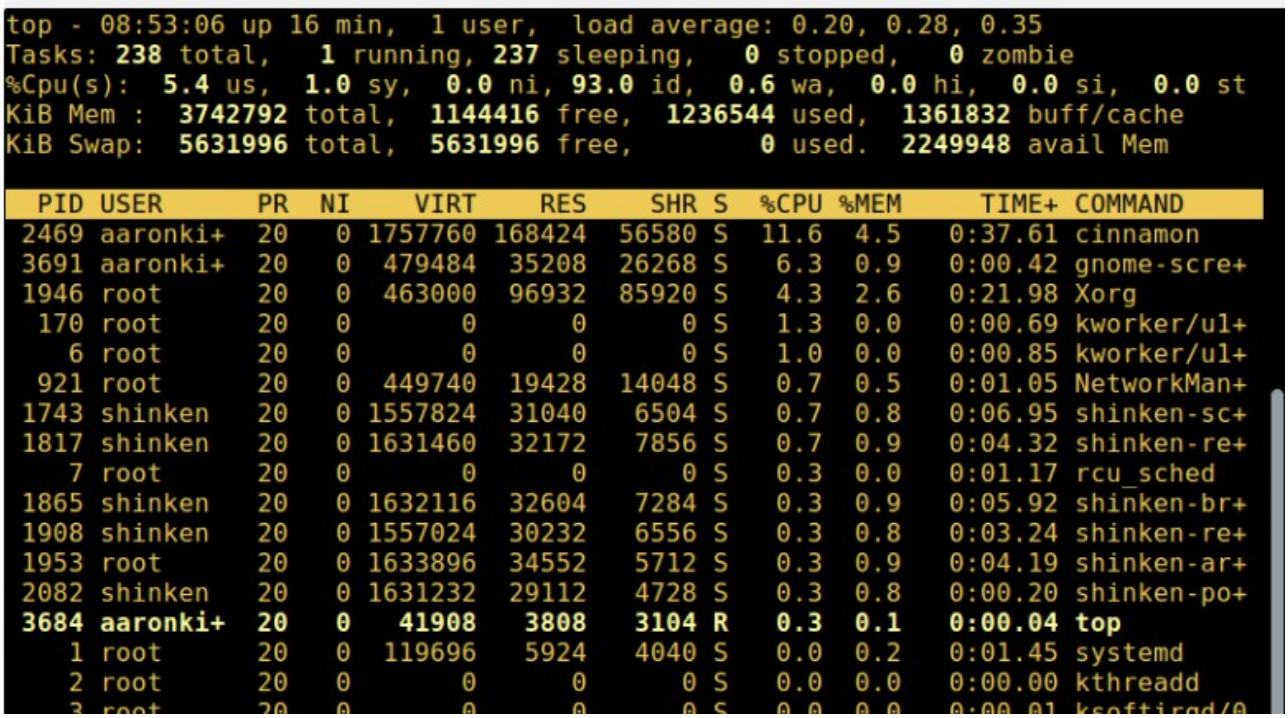
- SIGHUP 1 – sent to a process when its controlling terminal is closed.
- SIGINT 2– sent to a process by its controlling terminal when a user interrupts the process by pressing `[Ctrl+C]`.
- SIGQUIT 3– sent to a process if the user sends a quit signal `[Ctrl+D]`.
- SIGKILL 9– this signal immediately terminates (kills) a process and the process will not perform any clean-up operations.
- SIGTERM 15 – this a program termination signal (kill will send this by default).
- SIGTSTP 20 – sent to a process by its controlling terminal to request it to stop (terminal stop); initiated by the user pressing `[Ctrl+Z]`.

Changing Linux Process Priority

On the Linux system, all active processes have a priority and certain nice value. Processes with higher priority will normally get more CPU time than lower priority processes.

However, a system user with root privileges can influence this with the nice and renicecommands.
From the output of the top command, the NI shows the process nice value:

```
$ top
```



Field	Description	Example 1	Example 2
PID	The process ID of each task	1525	961
User	The username of task owner	Home	Root
PR	Priority Can be 20(highest) or -20(lowest)	20	20
NI	The nice value of a task	0	0
VIRT	Virtual memory used (kb)	1775	75972
RES	Physical memory used (kb)	100	51
SHR	Shared memory used (kb)	28	7952
S	Status There are five types: 'D' = uninterruptible sleep 'R' = running 'S' = sleeping 'T' = traced or stopped 'Z' = zombie	S	R

Field	Description	Example 1	Example 2
%CPU	% of CPU time	1.7	1.0
%MEM	Physical memory used	10	5.1
TIME+	Total CPU time	5:05.34	2:23.42
Command	Command name	Photoshop.exe	Xorg

NICE

Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.
To avoid this, you can tell your machine to prioritize processes as per your requirements. This priority is called Niceness in Linux, and it has a value between -20 to 19. The lower the Niceness index, the higher would be a priority given to that task.
The default value of all the processes is 0.
To start a process with a niceness value other than the default value use the following syntax

```
nice -n 'Nice value' process name
```

```
home@VirtualBox:~$ nice -n 19 banshee
```

If there is some process already running on the system, then you can 'Renice' its value using syntax.

```
renice 'nice value' -p 'PID'
```

To change Niceness, you can use the 'top' command to determine the PID (process id) and its Nice value. Later use the renice command to change the value.

Let us understand this by an example.

Checking the niceness value of the process 'banshee'

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 3293 home      20   0  277m   64m   35m S  96.4   6.4   9:56.72 banshee
```

Renicing the value to -20

```
home@VirtualBox:~$ sudo renice -20 -p 3293
[sudo] password for home:
3293 (process ID) old priority 0, new priority -20
```

The value changed to -20

```
 3293 home      0 -20  277m   64m   35m S  95.2   6.4   3:32.95 banshee
```