

Priority and nice values in Linux

```
top - 10:01:52 up 2:17, 0 users, load average: 5.06, 5.04, 5.00
Tasks: 7 total, 6 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 74.3 us, 0.2 sy, 25.4 ni, 0.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2046748 total, 76624 free, 256748 used, 1713376 buff/cache
KiB Swap: 1048572 total, 1048572 free, 0 used. 1626416 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
47	root	20	0	4624	864	800	R	99.3	0.0	55:25.71	sh
40	root	20	0	4624	832	768	R	99.0	0.0	61:28.80	sh
48	root	20	0	4624	888	820	R	98.7	0.0	55:21.20	sh
38	root	30	10	4624	780	712	R	59.8	0.0	35:45.61	sh
36	root	30	10	4624	864	800	R	41.2	0.0	34:54.98	sh
1	root	20	0	18504	3332	2908	S	0.0	0.2	0:00.11	bash
51	root	20	0	36640	3276	2800	R	0.0	0.2	0:00.85	top

How does Linux do memory management to execute your processes is a little complicated thing to explain. But luckily all the abstracted details around memory management like memory allocation and context switching b/w processes are very well optimized and developers don't have to worry about these things while writing high-level code.

What we should generally take into consideration is the amount of CPU bandwidth that our processes are eligible for, especially in cases of contention with other processes sharing the processing power of the CPU.

Before we jump into the priority and nice values and how they help us; It is important to have a top-level understanding of how CPUs run multiple processes at the same time. Basically, if a CPU has n cores then it can only

execute n processes in parallel. When the number of processes is more than n , the processes are switched between very fast using advanced context-switching mechanisms to provide multitasking. This lets us run multiple processes on a CPU even if it has just 1 core.

Why worry about the priority of your process?

Some processes may be highly CPU-intensive but not as important as others and hence can have a lower priority while others may or may not be highly CPU-intensive but are very important and hence should have higher priority. For example- if there is a process **A**, which detects fraud with input data and there is another process **B**, which makes hourly backups of some data, then the $\text{priority}(\text{A}) > \text{priority}(\text{B})$! This ensures that if both A and B are running at the same time, A would be allocated more processing bandwidth.

Now that we have enough context, let's dive into the specifics.

Priority value — The priority value is the process's actual priority which is used by the Linux kernel to schedule a task. In Linux system priorities are 0 to 139 in which 0 to 99 for real-time and 100 to 139 for users.

Nice value — Nice values are user-space values that we can use to control the priority of a process. The nice value range is -20 to +19 where -20 is highest, 0 default and +19 is lowest.

The relation between nice value and priority is as such -

```
Priority_value = Nice_value + 20
```

To see how these works together let us take a process that takes a lot of processing power continuously. I'll use a shell script (*infinite.sh*) which has an infinite loop in it to demonstrate how this works.

```
cat > infinite.sh
#!/bin/bash
for (( ; ; ))
do
    continue
done
```

We'll run this on a single core CPU for easy understanding & use **top command** (a program that periodically displays a sorted list of system processes and their details like pid, priority value, nice value, CPU usage, etc.) to monitor processes. The output for top before running infinite.sh:

Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie										
%Cpu(s): 0.0 us, 0.2 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st										
KiB Mem : 2046748 total, 107692 free, 259316 used, 1679740 buff/cache										
KiB Swap: 1048572 total, 1048572 free, 0 used. 1624196 avail Mem										
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
1	root	20	0	18504	3392	2976	S	0.0	0.2	0:00.03 bash
15	root	20	0	36612	3072	2624	R	0.0	0.2	0:00.00 top

Output of top command before running infinite.sh

Once we run infinite.sh in the background using `sh infinite.sh &` we see that this process (*PID-28*) is taking 100% processing power of the CPU.

```
top - 21:53:54 up 12:52, 0 users, load average: 0.73, 0.24, 0.08
Tasks: 3 total, 2 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 25.1 us, 0.1 sy, 0.0 ni, 74.8 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2046748 total, 109100 free, 256736 used, 1680912 buff/cache
KiB Swap: 1048572 total, 1048572 free, 0 used. 1626764 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
28	root	20	0	4624	884	816	R	100.0	0.0	0:58.28	sh
1	root	20	0	18504	3420	2976	S	0.0	0.2	0:00.05	bash
29	root	20	0	36612	3164	2716	R	0.0	0.2	0:00.01	top

If I run two more processes of infinite.sh then all of them (*PID-28,30,32*) get equal CPU as all have the same priority.

```
top - 21:57:19 up 12:55, 0 users, load average: 2.10, 1.11, 0.45
Tasks: 5 total, 4 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 25.2 us, 0.0 sy, 0.0 ni, 74.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2046748 total, 108480 free, 256904 used, 1681364 buff/cache
KiB Swap: 1048572 total, 1048572 free, 0 used. 1626620 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
30	root	20	0	4624	808	744	R	33.7	0.0	1:30.78	sh
28	root	20	0	4624	884	816	R	33.3	0.0	2:48.69	sh
32	root	20	0	4624	868	800	R	33.3	0.0	0:03.43	sh
1	root	20	0	18504	3420	2976	S	0.0	0.2	0:00.06	bash
33	root	20	0	36612	3252	2808	R	0.0	0.2	0:00.00	top

Now let us give these processes different nice values. There are two ways to do this:

1. Start the process with the nice value in the command as

```
nice -n nice_val [command]
```

For us, it could be something like `nice -n 10 sh infinite.sh &`

When we run this we get another process (*PID-34*) with the priority as **30** (as priority = 20 + nice_value). As it has the least priority, it gets the least amount of CPU.

```
top - 22:02:58 up 13:01, 0 users, load average: 3.28, 2.47, 1.27
Tasks: 6 total, 5 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 24.1 us, 0.1 sy, 0.9 ni, 74.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2046748 total, 106652 free, 257952 used, 1682144 buff/cache
KiB Swap: 1048572 total, 1048572 free, 0 used. 1625576 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
28	root	20	0	4624	884	816	R	32.3	0.0	4:41.53	sh
30	root	20	0	4624	808	744	R	32.3	0.0	3:23.62	sh
32	root	20	0	4624	868	800	R	32.0	0.0	1:56.27	sh
34	root	30	10	4624	804	740	R	3.7	0.0	0:00.76	sh
1	root	20	0	18504	3420	2976	S	0.0	0.2	0:00.06	bash
35	root	20	0	36612	3204	2760	R	0.0	0.2	0:00.00	top

2. Change the nice value of a running process using its PID

using `renice` as `renice -n nice_val -p [pid]`

For us, it could be something like `renice -n 5 -p 28`

When we run this, the process with *PID-28* gets its priority set from 20 to 25 and the CPU is allocated accordingly.

```
top - 22:12:36 up 13:11, 0 users, load average: 4.00, 3.81, 2.57
Tasks: 6 total, 5 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 20.6 us, 0.2 sy, 4.4 ni, 74.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2046748 total, 106004 free, 257492 used, 1683252 buff/cache
KiB Swap: 1048572 total, 1048572 free, 0 used. 1626020 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
30	root	20	0	4624	808	744	R	41.0	0.0	6:44.49	sh
32	root	20	0	4624	868	800	R	41.0	0.0	5:17.14	sh
28	root	25	5	4624	884	816	R	13.3	0.0	7:15.03	sh
34	root	30	10	4624	804	740	R	4.3	0.0	0:22.34	sh
1	root	20	0	18504	3420	2976	S	0.0	0.2	0:00.06	bash
37	root	20	0	36612	3184	2740	R	0.0	0.2	0:00.01	top

Note: Only root user can set the nice value from -20 to 19. Other users can only set nice values from 0 to 19.

You can also use `renice` to set nice values to all processes by a user using `renice -n nice_val -u [user]`