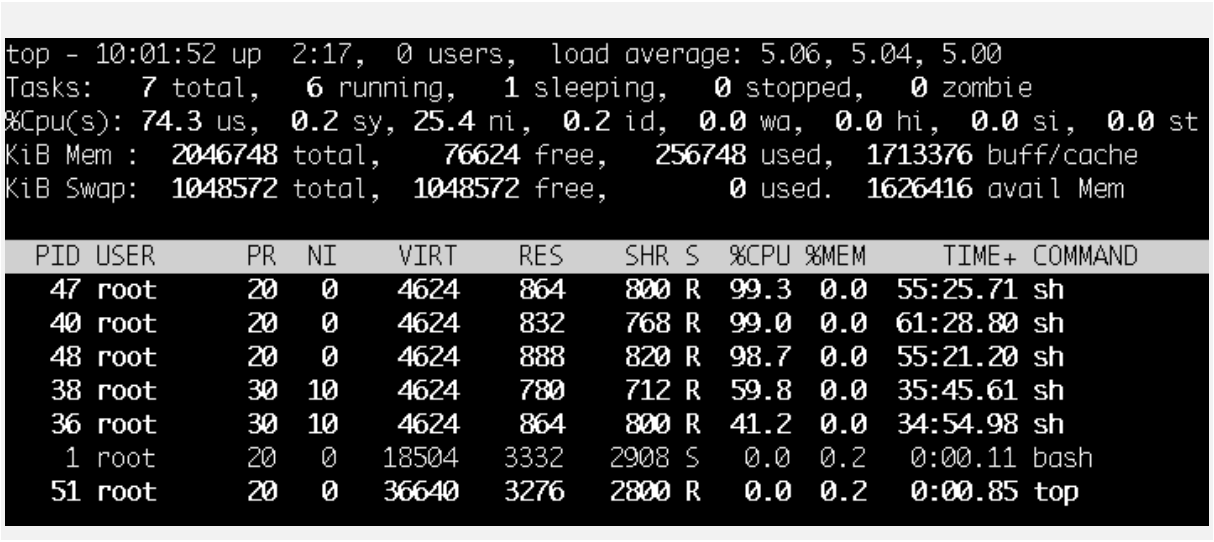
Priority and nice values in Linux



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 priority(A) > priority(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.

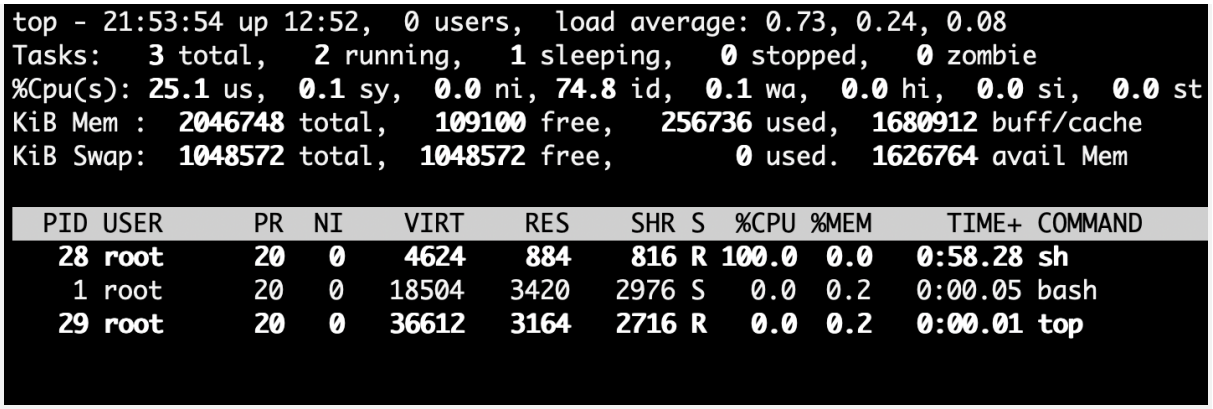
```
cat > infinite.sh
#! /bin/bash
for (( ; ; ))
do
 continue
done
```

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:

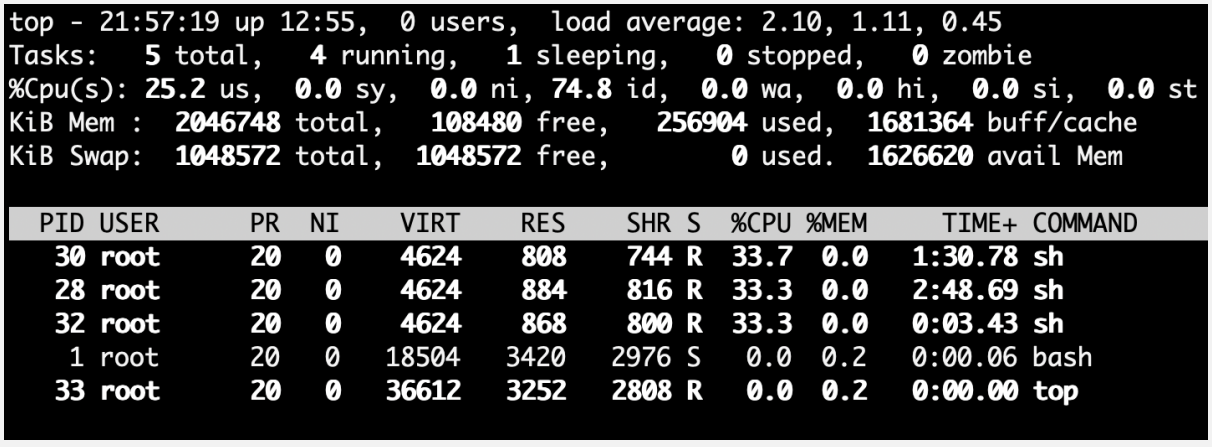
```
Tasks:   2 total,   1 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2046748 total,   107692 free,   259316 used,  1679740 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  1624196 avail Mem

  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.



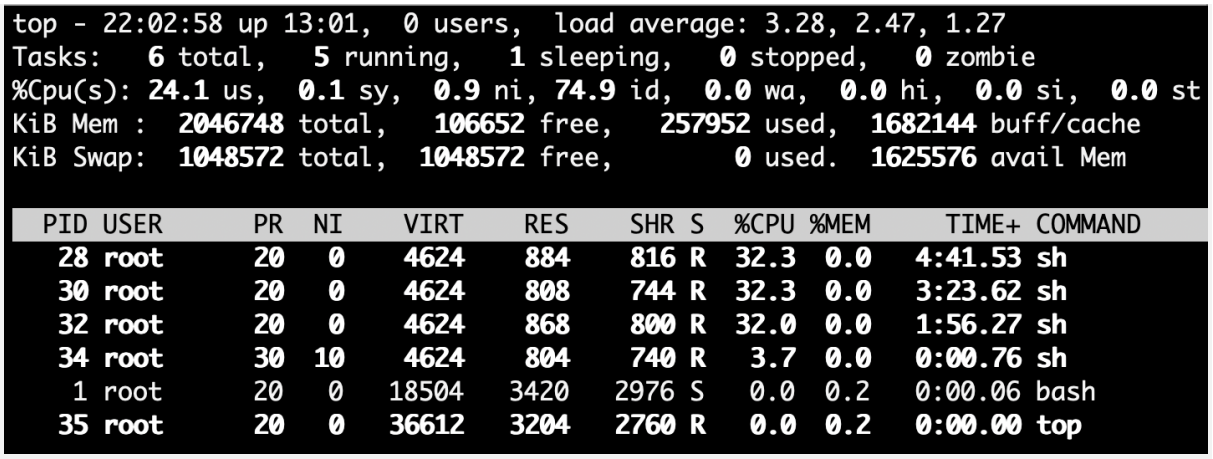
If I run two more processes of infinite.sh then all of them *(PID-28,30,32)* get equal CPU as all have the same priority.



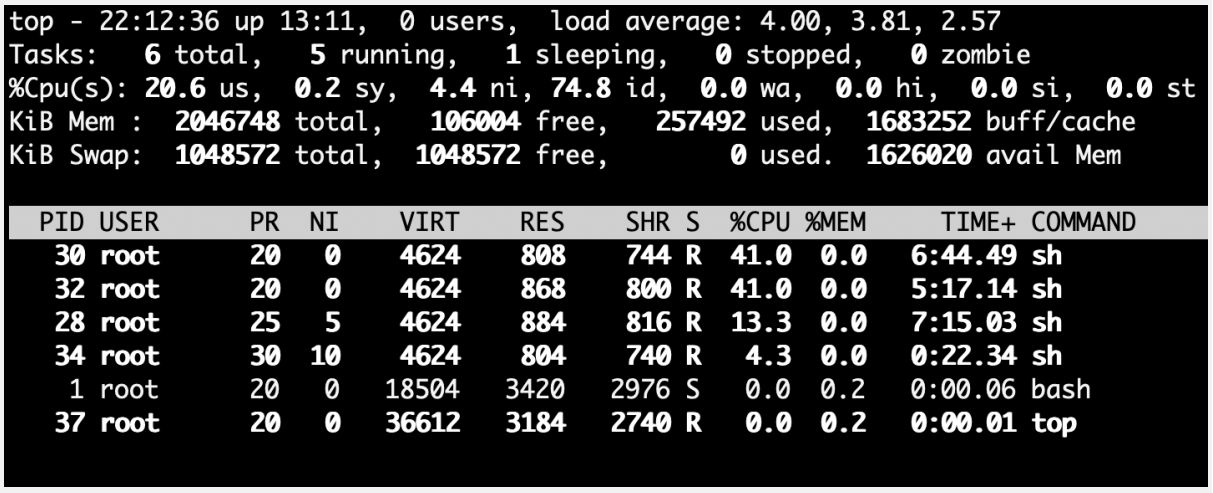
Now let us give these processes different nice values. There are two ways to do this:
1. Start the process with the nice value in the command as

```
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.



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.



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]`