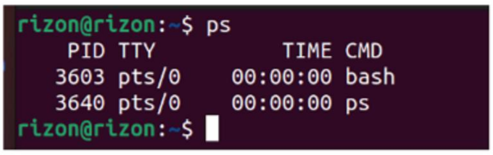
Processes In Linux - 1

* **Process (PS) :** They are just programs that are running on the machine, these process are generally managed by kernal and each and every process has a process ID (PID) associated with it.



```
rizon@rizon:~$ ps
    PID TTY          TIME CMD
   3603 pts/0    00:00:00 bash
   3640 pts/0    00:00:00 ps
rizon@rizon:~$ 
```

① PID ⇒ Process ID

② TTY ⇒ Controlling terminal of the process and each and every process are associated to a terminal.

③ Time ⇒ Total CPU usage time

④ Cmd ⇒ command.

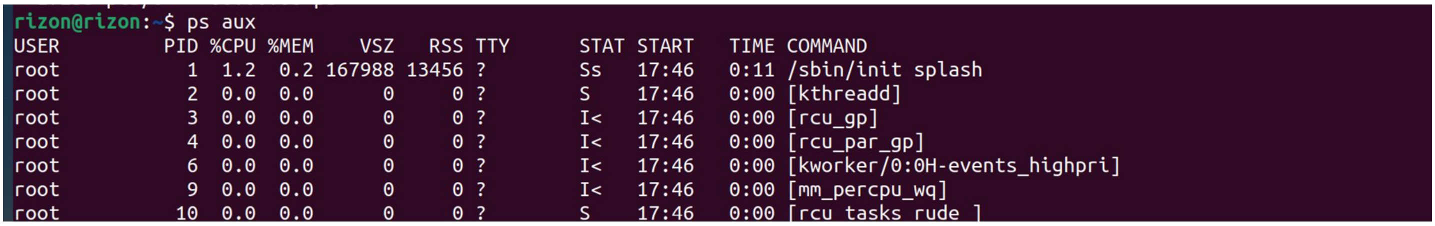
* **ps aux :**

It has 3 flag : a, u, x

a ⇒ All the process

u ⇒ Details about the process

x ⇒ list down all the process even the ones that don't have a terminal associated with them.



```
rizon@rizon:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  1.2  0.2 167988 13456 ?        Ss   17:46   0:11 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    17:46   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   17:46   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   17:46   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   17:46   0:00 [kworker/0:0H-events_highpri]
root           9  0.0  0.0      0     0 ?        I<   17:46   0:00 [mm_percpu_wq]
root          10  0.0  0.0      0     0 ?        S    17:46   0:00 [rcu_tasks_rude_]
```

ps aux

(1) User

(2) PID

(3) % CPU ⇒ how much % of the CPU is the system currently using.

(4) % mem ⇒ % of memory that the process is using out of the entire physical memory,

(5) VSZ ⇒ Virtual memory usage of the entire process (Reserved memory)

Maximum memory available ⇒ kb format

(6) RSS ⇒ Memory currently used.

$\boxed{VSZ >= RSS}$

(7) TTY ⇒ It has three value

- ? ⇒ No terminal associated
- tty
- pts

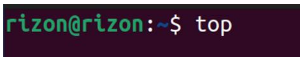
(8) STAT ⇒ Status of the process.

(9) START ⇒ What time it started the process

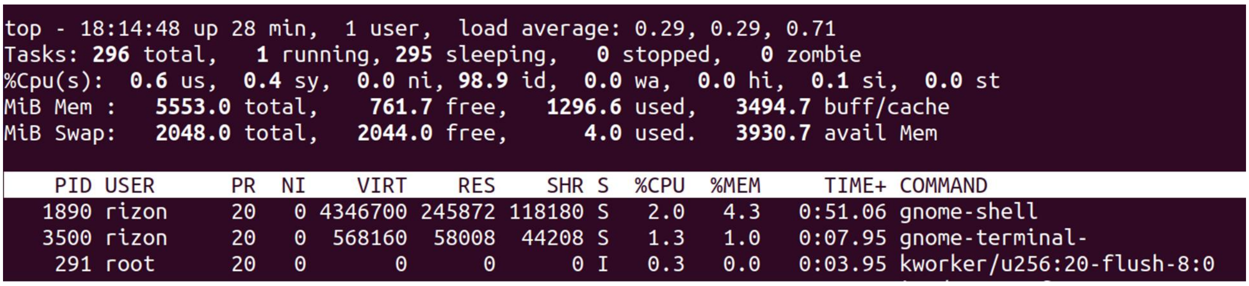
(10) TIME

(ii) COMMAND

* top : kind of psx version, it gives the real time data.



```
rizon@rizon:~$ top
```



```
top - 18:14:48 up 28 min,  1 user,  load average: 0.29, 0.29, 0.71
Tasks: 296 total,   1 running, 295 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.6 us,  0.4 sy,  0.0 ni, 98.9 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :   5553.0 total,    761.7 free,   1296.6 used,   3494.7 buff/cache
MiB Swap:   2048.0 total,   2044.0 free,      4.0 used.   3930.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1890 rizon     20   0 4346700 245872 118180 S   2.0   4.3   0:51.06 gnome-shell
   3500 rizon     20   0  568160  58008  44208 S   1.3   1.0   0:07.95 gnome-terminal-
    291 root      20   0       0      0      0 I   0.3   0.0   0:03.95 kworker/u256:20-flush-8:0
```

- * Controlling Terminal :

① TTY1 => Regular Terminal

* No graphics associated with it, it simple a shell only use keyboard to type commands

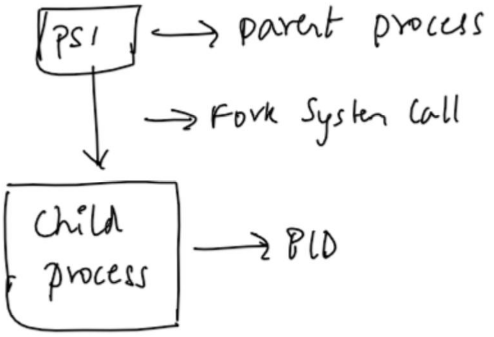
② pts/1 => Pseudo Terminal

Process are an instance of a program

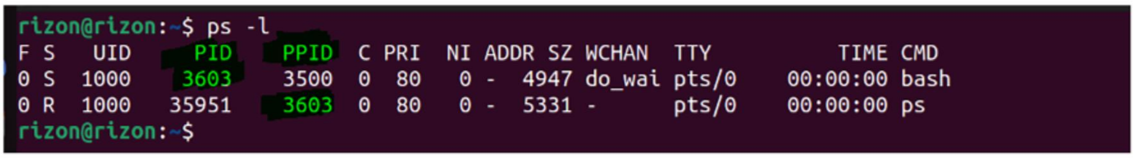
Who manage the processes?

Kernal it's responsible for all the processes that are there and when it run any program/command it's the kernal who loads up the code of that program. in the memory then determines and allocates all the resources that the program may need and it's generally keeps a tap on all the processes that are running it.

How is a process created?

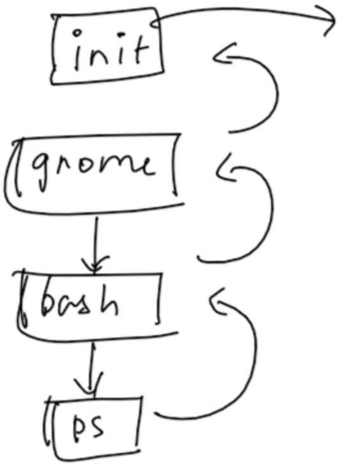


execve system call



```
rizon@rizon:~$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    3603    3500  0  80   0 -  4947 do_wai pts/0    00:00:00 bash
0 R  1000   35951    3603  0  80   0 -  5331 -      pts/0    00:00:00 ps
rizon@rizon:~$
```

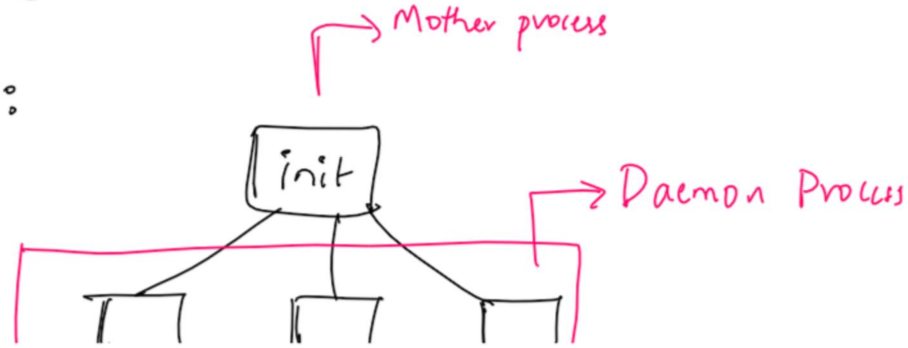
Mother process:

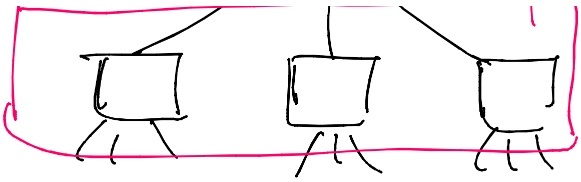


It's known as Mother process

It has PID of '1'.

Daemon Process:





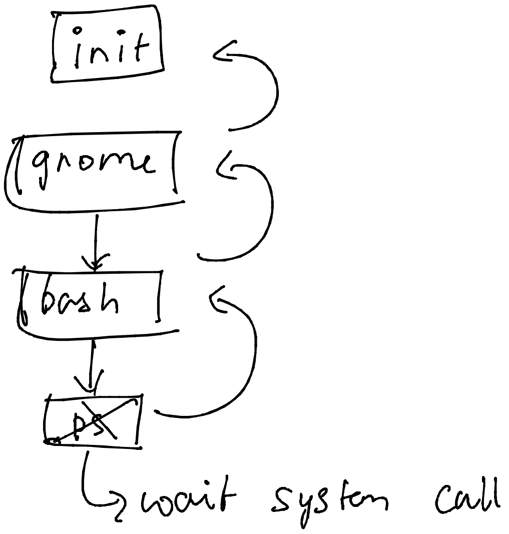
This are the process is responsible for keeping the system running.

* Terminate a process:

- Exist system call
- Termination status

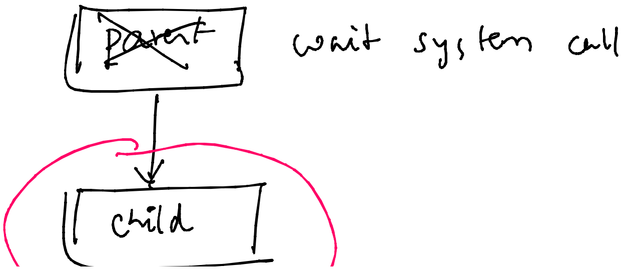
↓

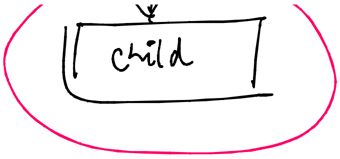
0 → process succeded



Orphan Cases :

If parent dies.





Orphan Cases

In this scenario the kernal knows that whenever this child process will terminate it doesn't have a parent i.e. going to use the weight system call to acknol. that their child has terminate.