

```
root@Biel: ~  
root@Biel:~# yes > /dev/null &  
yes > /dev/null &  
[1] 450  
[2] 451  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# ps -eo pid,comm,pri,nice | grep yes  
450 yes 19 0  
451 yes 19 0  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# renice 10 -p 450  
450 (process ID) old priority 0, new priority 10  
root@Biel:~# |
```

Os comandos, “**yes > /dev/null &**” funcionam assim:

O comando “**yes**” gera uma saída infinita de “**y**”. (significa que ele imprime a letra “y” de forma contínua, sem parar).

Como a saída é redirecionada para “**/dev/null**”, ele apenas consome CPU sem imprimir nada.

O “**&**” executa os processos em segundo plano.

Nesse caso gerou os processos **450** e **451**.

Depois utilizei o comando “**ps -eo pid,comm,pri,nice | grep yes**” para listar os 2 processos criados da seguinte forma:

Vou fracionar o comando e explicar como ele funciona:

\***ps** : Exibe informações sobre os processos em execução no sistema.

\***-e** : Essa opção diz ao ps para listar todos os processos ativos no sistema, independentemente de qual terminal eles estejam associados.

\***-o** : Essa opção permite que você defina quais colunas serão exibidas na saída do comando. Por exemplo, usando -o pid,comm,pri,nice, você especifica que deseja ver as colunas:

**pid**: O identificador único do processo.

**comm**: O nome do comando ou programa que está sendo executado.

**pri**: A prioridade interna do processo definida pelo kernel.

**nice**: Um valor ajustável que influencia a prioridade de CPU do processo.

o **grep** para filtrar e mostrar apenas as linhas que contêm a palavra "**yes**", facilitando a identificação dos processos relacionados ao comando **yes**.

O Comando “**renice 10 -p <PID>**” funciona da seguinte forma:

O **renice** ajusta o valor de "nice" do processo identificado por <PID>, tornando-o menos prioritário (ou seja, ele recebe menos tempo de CPU).

No caso tornando o processo **450** menos prioritário.

```
root@Biet: ~
top - 16:52:12 up 7 min, 1 user, load average: 1.56, 1.19, 0.58
Tasks: 25 total, 3 running, 22 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.5 us, 7.4 sy, 2.5 ni, 87.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7788.3 total, 7061.9 free, 645.4 used, 310.3 buff/cache
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used, 7142.9 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
  450 root        30  10   3124  1064   976 R 100.0   0.0   0:52.71 yes
  451 root        20   0   3124  1084   996 R 100.0   0.0   0:52.71 yes
    1 root        20   0  21508 12856  9592 S   0.0   0.2   0:00.29 systemd
    2 root        20   0   2776  1920  1796 S   0.0   0.0   0:00.00 init-systemd(Ub
    6 root        20   0   2776    68    68 S   0.0   0.0   0:00.00 init
   48 root        19  -1  34060 12544 11452 S   0.0   0.2   0:00.06 systemd-journal
   93 root        20   0  24124  6136  4780 S   0.0   0.1   0:00.07 systemd-udev
  103 systemd+    20   0  21452 11848  9656 S   0.0   0.1   0:00.05 systemd-resolve
  104 systemd+    20   0  91020  6604  5748 S   0.0   0.1   0:00.04 systemd-timesyn
  171 root        20   0   4236  2608  2364 S   0.0   0.0   0:00.00 cron
  172 message+    20   0   9532  5156  4528 S   0.0   0.1   0:00.02 dbus-daemon
  179 root        20   0  17976  8236  7216 S   0.0   0.1   0:00.05 systemd-logind
  200 root        20   0   3160  1084   992 S   0.0   0.0   0:00.00 agetty
  204 syslog      20   0  222508 5012  4420 S   0.0   0.1   0:00.03 rsyslogd
  207 root        20   0   3116  1192  1108 S   0.0   0.0   0:00.00 agetty
  208 root        20   0  107012 22164 12828 S   0.0   0.3   0:00.08 unattended-upgr
  289 root        20   0   6668  4588  3820 S   0.0   0.1   0:00.00 login
  333 root        20   0  20260 11312  9236 S   0.0   0.1   0:00.04 systemd
  334 root        20   0   21152 1708    0 S   0.0   0.0   0:00.00 (sd-pam)
  345 root        20   0   5940  5120  3596 S   0.0   0.1   0:00.00 bash
  379 root        20   0 1756096 15820  9340 S   0.0   0.2   0:00.03 wsl-pro-service
  427 root        20   0   2780   208    80 S   0.0   0.0   0:00.00 SessionLeader
  428 root        20   0   2780   212    80 S   0.0   0.0   0:00.00 Relay(430)
  430 root        20   0   6072  5180  3504 S   0.0   0.1   0:00.01 bash
  455 root        20   0   9324  5164  3000 R   0.0   0.1   0:00.00 top
```

Aqui foi utilizado o comando “**top**” para identificar a prioridade dos processos em aberto.

Podemos visualizar que o processo **450** possui agora um **NI** de **10** adicionado a pouco. Isso indica que agora ele tem menor prioridade em comparação a processos com NI ou nice menor.

```
root@Biel: ~  
451 root      20    0      3124    1084    996 R 100.0  0.0   0:58.89 yes  
1 root        20    0      21508   12856   9592 S  0.0   0.2   0:00.29 systemd  
2 root        20    0      2776    1920   1796 S  0.0   0.0   0:00.00 init-systemd(Ub  
6 root        20    0      2776     68    68 S  0.0   0.0   0:00.00 init  
48 root       19   -1     34060  12544   11452 S  0.0   0.2   0:00.06 systemd-journal  
93 root       20    0      24124    6136   4780 S  0.0   0.1   0:00.07 systemd-udev  
103 systemd+  20    0      21452   11848   9656 S  0.0   0.1   0:00.05 systemd-resolve  
104 systemd+  20    0      91020    6604   5748 S  0.0   0.1   0:00.04 systemd-timesyn  
171 root      20    0      4236    2608   2364 S  0.0   0.0   0:00.00 cron  
172 message+  20    0      9532    5156   4528 S  0.0   0.1   0:00.02 dbus-daemon  
179 root      20    0      17976    8236   7216 S  0.0   0.1   0:00.05 systemd-logind  
200 root      20    0      3160    1084    992 S  0.0   0.0   0:00.00 agetty  
204 syslog    20    0      222508   5012   4420 S  0.0   0.1   0:00.03 rsyslogd  
207 root      20    0      3116    1192   1108 S  0.0   0.0   0:00.00 agetty  
208 root      20    0      107012  22164  12828 S  0.0   0.3   0:00.08 unattended-upgr  
289 root      20    0      6668    4588   3820 S  0.0   0.1   0:00.00 login  
333 root      20    0      20260   11312   9236 S  0.0   0.1   0:00.04 systemd  
334 root      20    0      21152   1708     0 S  0.0   0.0   0:00.00 (sd-pam)  
345 root      20    0      5940    5120   3596 S  0.0   0.1   0:00.00 bash  
379 root      20    0      1756096 15820   9340 S  0.0   0.2   0:00.03 wsl-pro-service  
427 root      20    0      2780     208     80 S  0.0   0.0   0:00.00 SessionLeader  
428 root      20    0      2780     212     80 S  0.0   0.0   0:00.00 Relay(430)  
430 root      20    0      6072    5180   3504 S  0.0   0.1   0:00.01 bash  
455 root      20    0      9324    5164   3000 R  0.0   0.1   0:00.00 top  
  
root@Biel:~# sleep 300 &  
[3] 456  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# kill 450  
root@Biel:~#  
[1] Terminated  
yes > /dev/null
```

Como o “**Sleep**” funciona:

O utilitário **sleep**, que simplesmente aguarda (dorme) por um determinado número de segundos. Neste caso, **300** segundos e mais uma vez utilizei o “**&**” para que o processo funcione em segundo plano, o que permite que eu continue usando o terminal.

Como o “**Kill**” funciona:

O comando **kill** envia um sinal para o processo especificado pelo **PID** que solicita que o processo finalize sua execução. No caso o processo **450**.

```
root@Biel: ~  
428 root      20    0   2780    212     80 S    0.0   0.0   0:00.00 ReLay(430)  
430 root      20    0   6072    5180   3504 S    0.0   0.1   0:00.01 bash  
455 root      20    0   9324    5164   3000 R    0.0   0.1   0:00.00 top  
  
root@Biel:~# sleep 300 &  
[3] 456  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# kill 450  
root@Biel:~#  
[1] Terminated                  yes > /dev/null  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# ps  
  PID TTY          TIME CMD  
  430 pts/0        00:00:00 bash  
  451 pts/0        00:03:07 yes  
  456 pts/0        00:00:00 sleep  
  457 pts/0        00:00:00 ps  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# free -h  
Mem:            7.6Gi       646Mi       6.9Gi       3.2Mi      buff/cache    310Mi      available  
Swap:           2.0Gi          0B       2.0Gi  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~#  
root@Biel:~# |
```

Como o “**free -h**” funciona:

Esse comando exibe o uso de memória do sistema, incluindo a memória total, usada, livre e a memória alocada para cache e buffers. A opção **-h** (human-readable) mostra os valores em formatos fáceis de entender (como MB ou GB), facilitando a comparação entre memória utilizada e livre.

Também reutilizei o comando ps, desta vez sem filtros para visualizar todos os processos em execução.

```
root@Biel: ~  
[3]+ Done sleep 300  
root@Biel:~# ^C  
root@Biel:~# pmap 451  
451: yes  
000055a190ebe000 8K r---- yes  
000055a190ec0000 16K r-x-- yes  
000055a190ec4000 4K r---- yes  
000055a190ec5000 4K r---- yes  
000055a190ec6000 4K rw--- yes  
000055a1b97bc000 132K rw--- [ anon ]  
00007f90c5144000 356K r---- LC_CTYPE  
00007f90c519d000 4K r---- LC_NUMERIC  
00007f90c519e000 4K r---- LC_TIME  
00007f90c519f000 4K r---- LC_COLLATE  
00007f90c51a0000 4K r---- LC_MONETARY  
00007f90c51a1000 4K r---- SYS_LC_MESSAGES  
00007f90c51a2000 4K r---- LC_PAPER  
00007f90c51a3000 28K r--s- gconv-modules.cache  
00007f90c51aa000 12K rw--- [ anon ]  
00007f90c51ad000 160K r---- libc.so.6  
00007f90c51d5000 1568K r-x-- libc.so.6  
00007f90c535d000 316K r---- libc.so.6  
00007f90c53ac000 16K r---- libc.so.6  
00007f90c53b0000 8K rw--- libc.so.6  
00007f90c53b2000 52K rw--- [ anon ]  
00007f90c53bf000 4K r---- LC_NAME  
00007f90c53c0000 4K r---- LC_ADDRESS  
00007f90c53c1000 4K r---- LC_TELEPHONE  
00007f90c53c2000 4K r---- LC_MEASUREMENT  
00007f90c53c3000 4K r---- LC_IDENTIFICATION  
00007f90c53c4000 8K rw--- [ anon ]  
00007f90c53c6000 4K r---- ld-linux-x86-64.so.2  
00007f90c53c7000 172K r-x-- ld-linux-x86-64.so.2  
00007f90c53f2000 40K r---- ld-linux-x86-64.so.2  
00007f90c53fc000 8K r---- ld-linux-x86-64.so.2  
00007f90c53fe000 8K rw--- ld-linux-x86-64.so.2  
00007fff9a1e9000 132K rw--- [ stack ]  
00007fff9a28a000 16K r---- [ anon ]  
00007fff9a28e000 8K r-x-- [ anon ]  
total 3124K  
root@Biel:~# |
```

Como o “**pmap <PID>**” funciona:

O comando **pmap** mostra o mapa de memória de um processo específico, identificado pelo **PID**. Ele lista as diferentes regiões de memória que o processo está utilizando, como áreas de código, dados, bibliotecas compartilhadas, e mais. Essa ferramenta é útil para analisar o impacto de um processo específico no uso de memória.

Como apresentado utilizei no processo **451**.