```
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 camando "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.

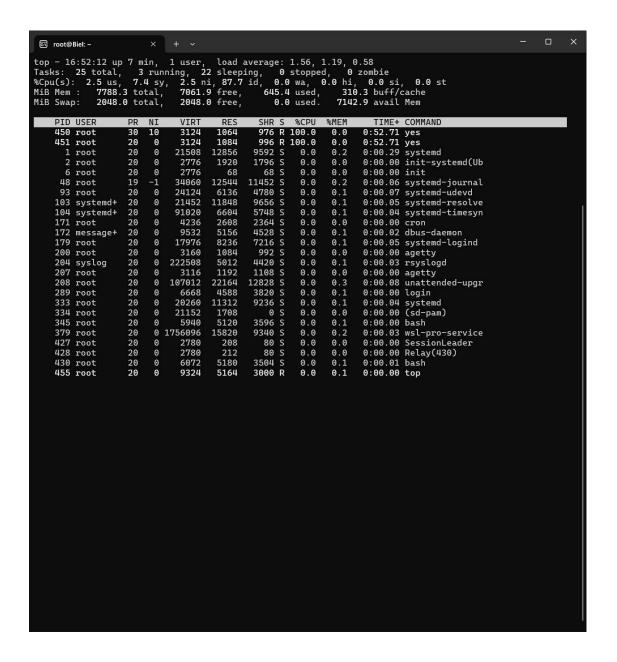
nice: Um valor ajustável que influencia a prioridade de CPU do processo.

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

O Camando "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).

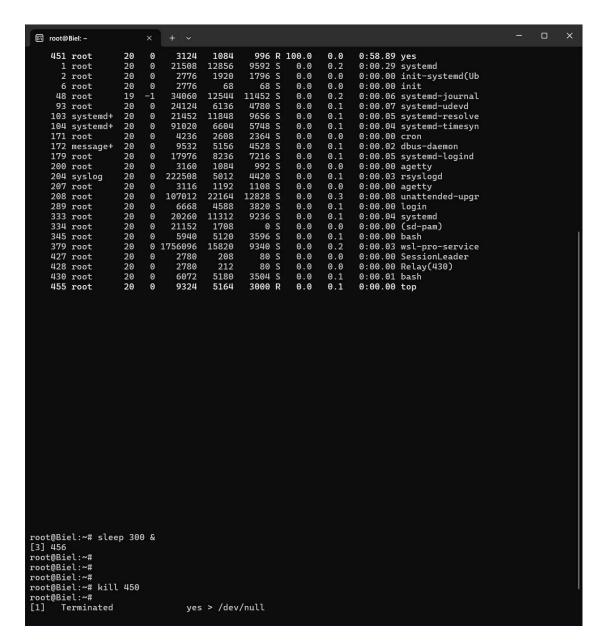
No caso tornando o processo 450 menos prioritário.



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.



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: ~
                                                                                                    0:00.00 Relay(430)
0:00.01 bash
0:00.00 top
                                                       212
5180
5164
                                            2780
      428 root
                            20
                                                                      80 S
                                            6072
9324
                                                                   3504 S
3000 R
      430 root
      455 root
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:~#
 root@Biel:~# ps
PID TTY
430 pts/0
                          TIME CMD
00:00:00 bash
00:03:07 yes
00:00:00 sleep
00:00:00 ps
      451 pts/0
456 pts/0
457 pts/0
root@Biel:~#
root@Biel:~#
root@Biel:~#
root@Biel:~# free -h
                       total
7.6Gi
2.0Gi
                                                              free
6.9Gi
2.0Gi
                                           used
646Mi
                                                                                shared buff/cache
3.2Mi 310Mi
                                                                                                                  available
Mem:
                                                                                                                        7.0Gi
Swap:
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
                             16K r-x--
4K r----
000055a190ec0000
                                           yes
000055a190ec4000
                                           yes
yes
000055a190ec5000
                                           yes
[ anon ]
LC_CTYPE
LC_NUMERIC
LC_TIME
LC_COLLATE
000055a190ec6000
                              4K rw-
000055a1b97bc000
                            132K rw-
00007f90c5144000
                            356K r
00007f90c519d000
                              4K r
00007f90c519e000
                              4K r
00007f90c519f000
                                           LC_MONETARY
SYS_LC_MESSAGES
LC_PAPER
00007f90c51a0000
                              4K r
00007f90c51a1000
                              4K r
00007f90c51a2000
                              4K r-
                             28K r
00007f90c51a3000
                                           gconv-modules.cache
                                           [ anon ]
00007f90c51aa000
                             12K rw---
00007f90c51ad000
                            160K r-
00007f90c51d5000
                          1568K r-x--
                                           libc.so.6
00007f90c535d000
00007f90c53ac000
                            316K r-
                                           libc.so.6
                                           libc.so.6
                             16K r
00007f90c53b0000
                                           libc.so.6
                                           [ anon ]
LC_NAME
LC_ADDRESS
00007f90c53b2000
00007f90c53bf000
                             52K rw---
                              4K r-
00007f90c53c0000
                                           LC_ADDRESS
LC_TELEPHONE
LC_MEASUREMENT
LC_IDENTIFICATION
[ anon ]
ld-Linux-x86-64.so.2
ld-Linux-x86-64.so.2
00007f90c53c1000
00007f90c53c2000
                              4K r-
00007f90c53c3000
                              8K rw-
00007f90c53c4000
00007f90c53c6000
00007f90c53c7000
                            172K r-x--
00007f90c53f2000
00007f90c53fc000
                                           ld-linux-x86-64.so.2
ld-linux-x86-64.so.2
ld-linux-x86-64.so.2
                             40K r-
                              8K r-
00007f90c53fe000
                              8K rw-
                                             [ stack ]
[ anon ]
[ anon ]
00007fff9a1e9000
                            132K rw---
00007fff9a28a000
                             16K r----
00007fff9a28e000
                              8K r-x--
                          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.