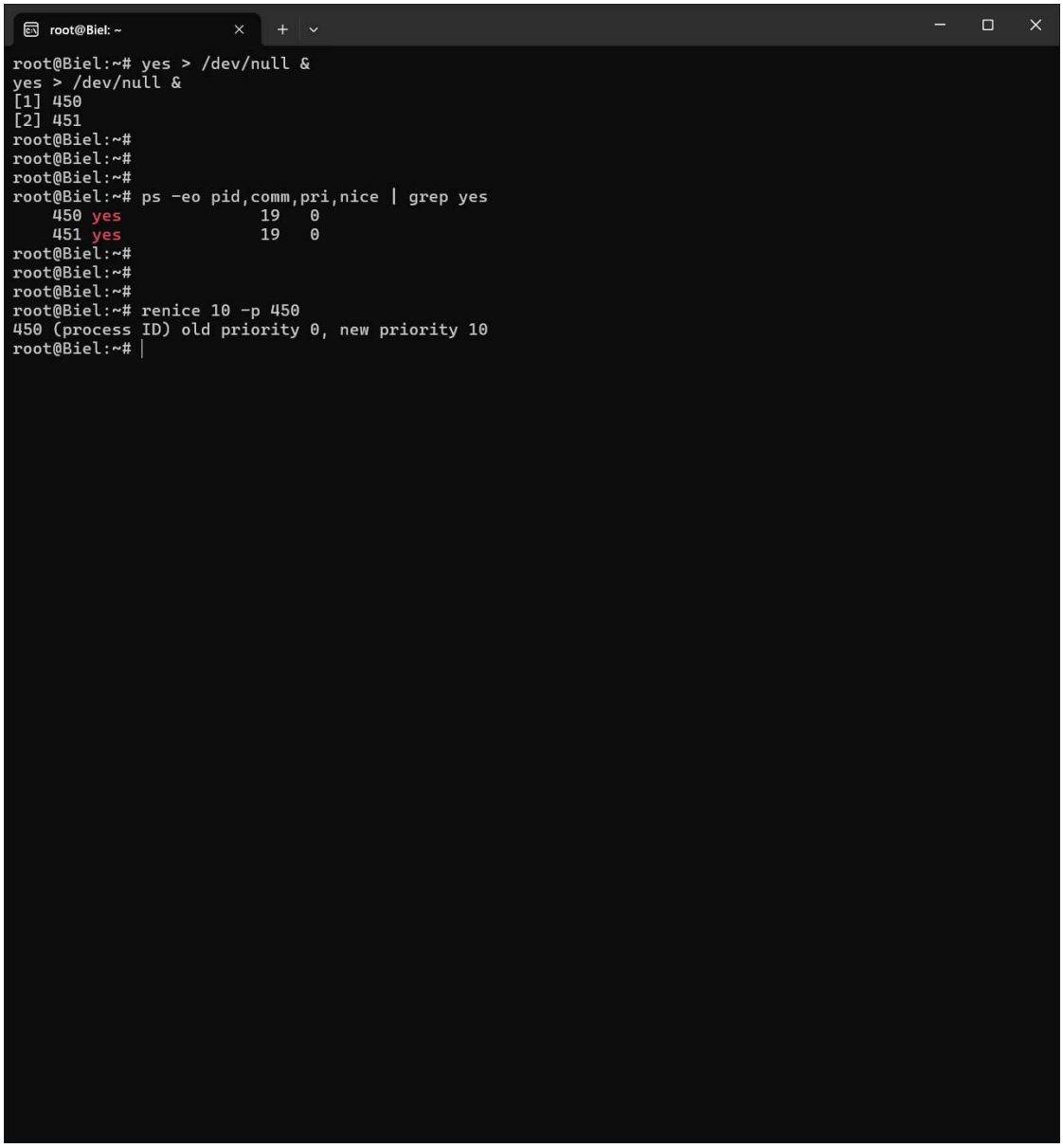


Os comandos, “**yes > /dev/null &**” funcionam assim:

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.

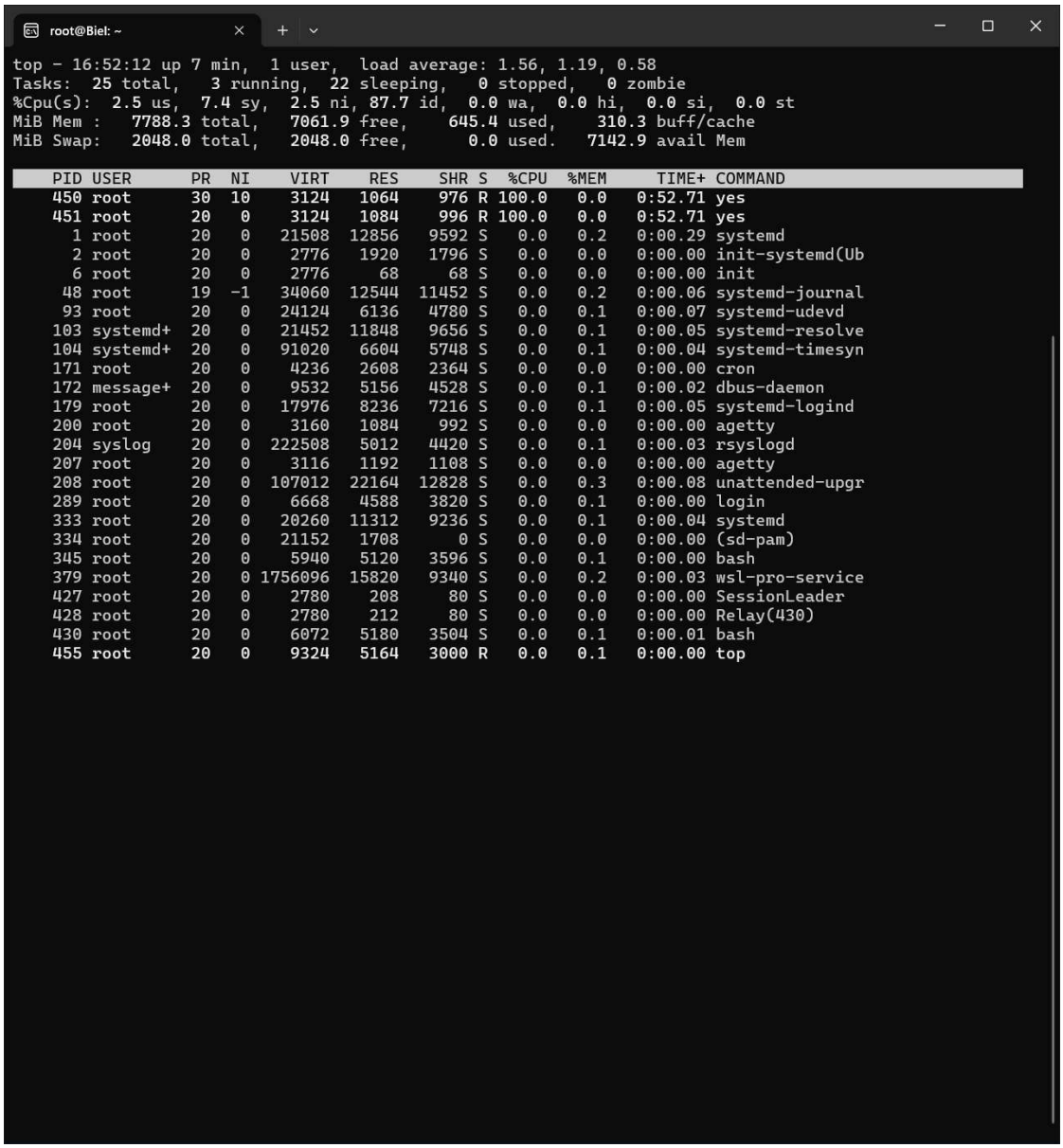
nice: Um valor ajustável que influencia a prioridade de CPU do processo.

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).

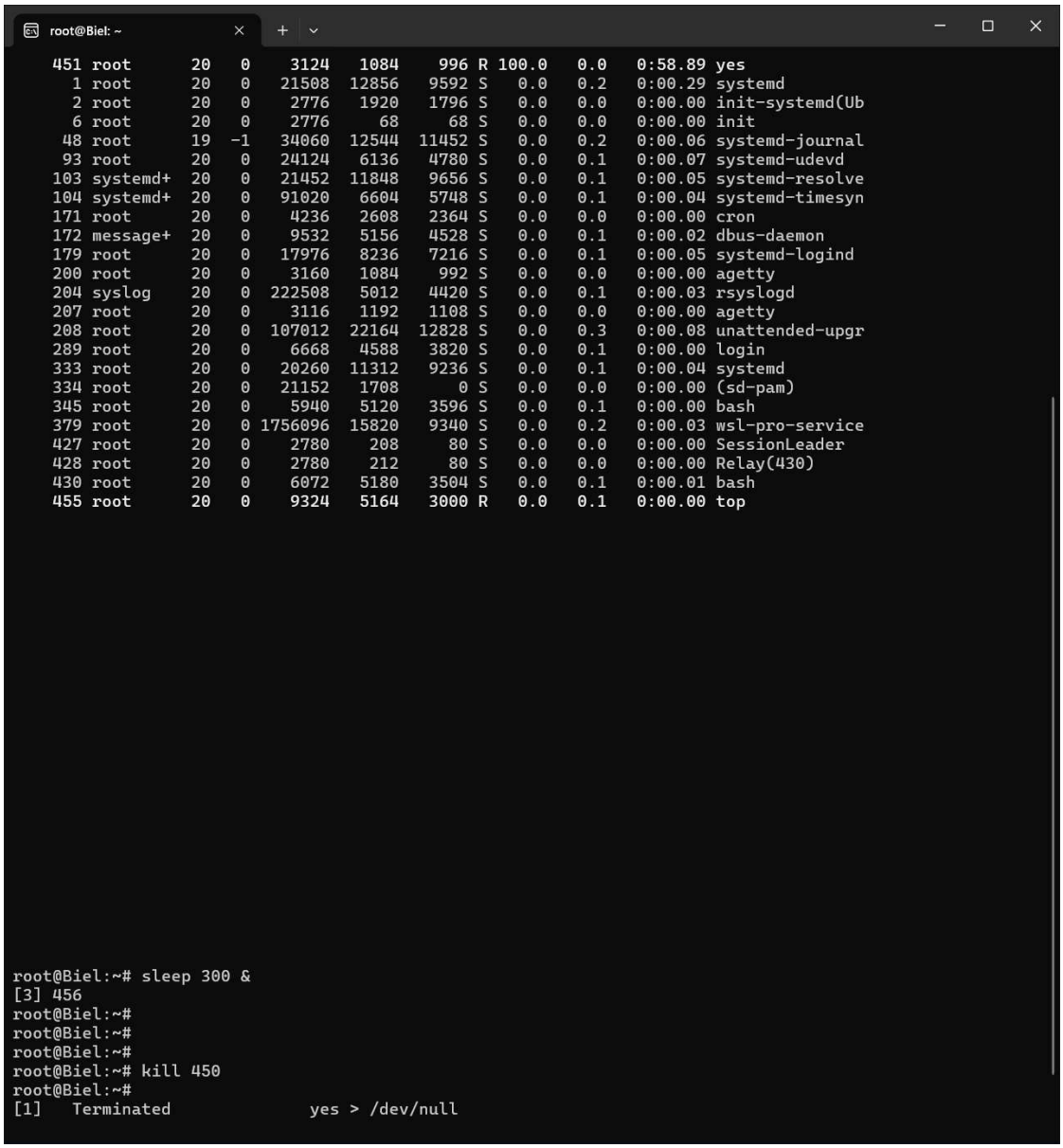
No caso tornando o processo **450** menos prioritário.



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.

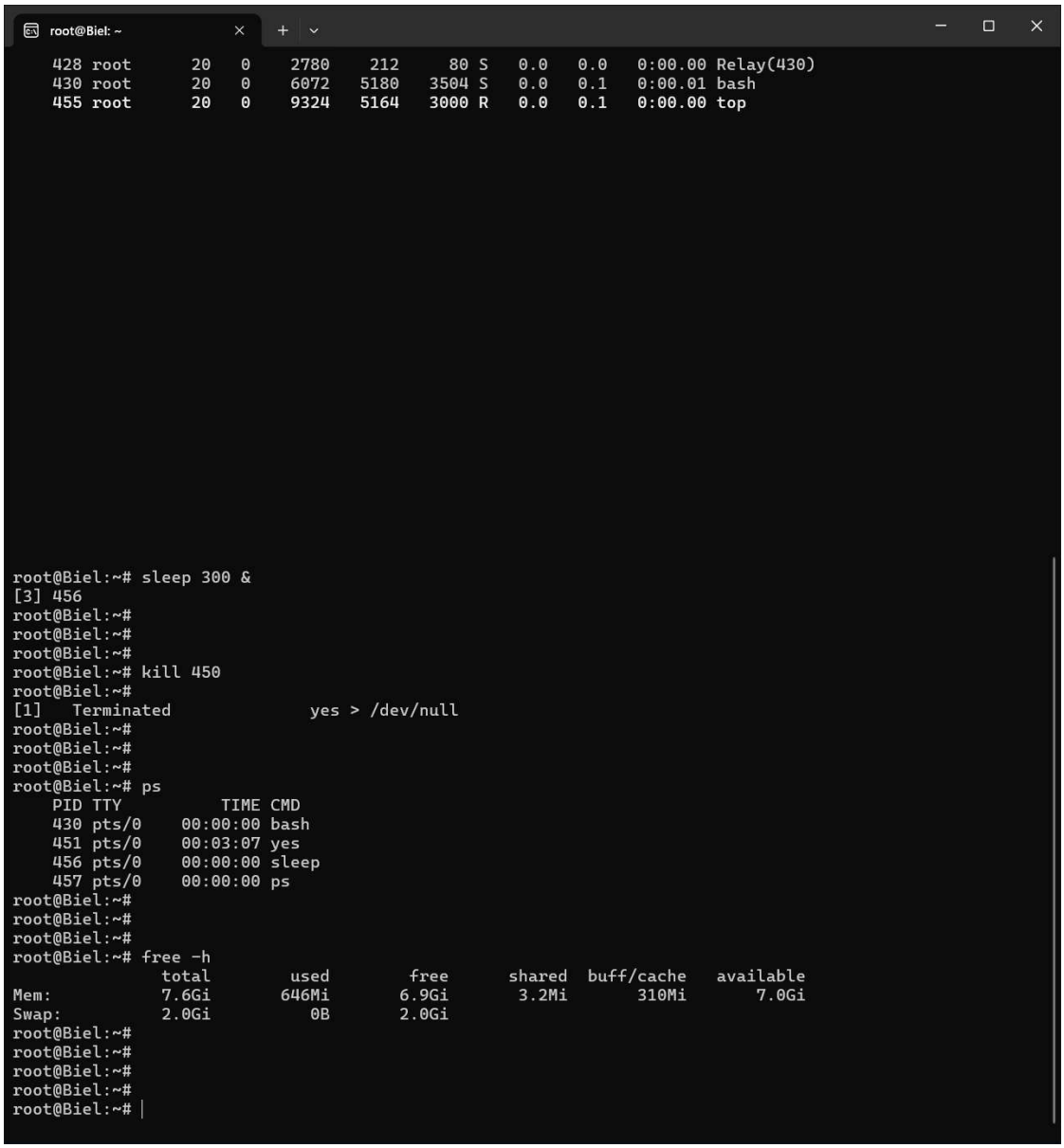


Como o “**Sleep**” funciona:

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:

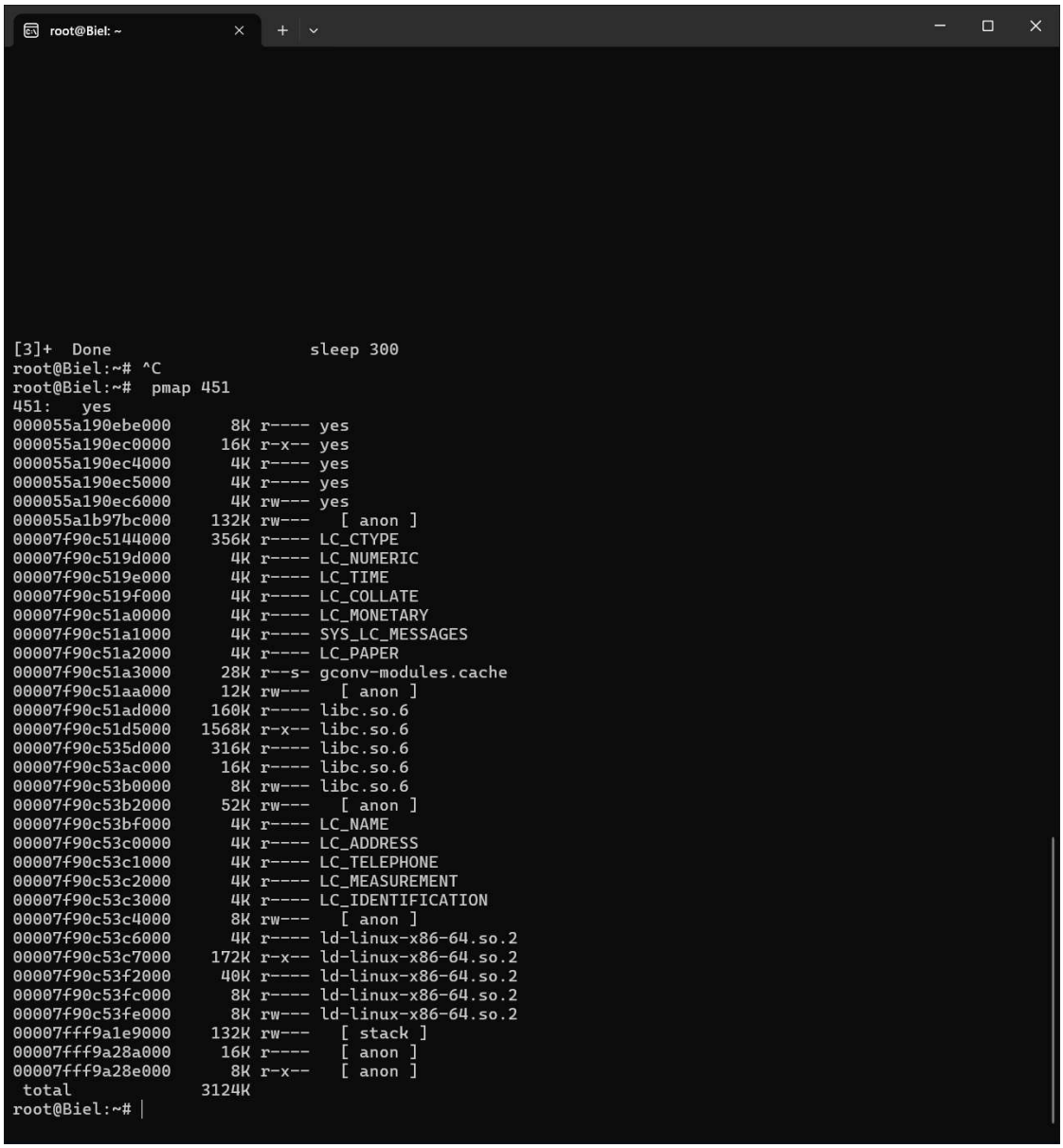
O comando **kill** envia um sinal para o processo especificado pelo **PID** que solicita que o processo finalize sua execução. No caso o processo **450.**



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.



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.**