

**Taxas de Leitura/Escrita de Processos em Bash**

**Trabalho realizado por:**

André Almeida Oliveira, nº107637

Duarte Carvalho da Cruz, nº 107359

**Sistemas Operativos**

**Prof.** José Nuno Panelas Nunes Lau

Ano Letivo 2022/2023

**Índice**

[Introdução 2](#_Toc120540357)

[Metodologia 3](#_Toc120540358)

[Primeira Parte 3](#_Toc120540359)

[1)Argumentos de entrada 3](#_Toc120540360)

[2)Inicialização de variáveis 4](#_Toc120540361)

[Segunda Parte 5](#_Toc120540362)

[1)CD /PROC 5](#_Toc120540363)

[2)listas para armazenamento de dados 5](#_Toc120540364)

[3)processos válidos 5](#_Toc120540365)

[4)Procura e armazenamento das informações de cada processo 7](#_Toc120540366)

[Terceira parte 8](#_Toc120540367)

[1)print da tabela 8](#_Toc120540368)

[2)Erros e exceções 9](#_Toc120540369)

[Validação da solução 10](#_Toc120540370)

[Erros e avisos 10](#_Toc120540371)

[TABELAS DE PROCESSOS 11](#_Toc120540372)

[Conclusão 14](#_Toc120540373)

\

# **Introdução**

Como objetivo deste projeto, tivemos de desenvolver um script em bash, denominado “rwstat.sh”, com o intuito de apresentar informações tabeladas dos processos ativos no nosso computador.

Após a execução deste script, como referido acima, serão apresentadas algumas estatísticas, tais como, o número de bytes lidos/escritos e a respetiva taxa de leitura/escrita, apresentada em bytes por segundo, assim como o respetivo nome do processo, o nome do utilizador que tem os processos ativos e a data de início dos mesmos.

Na execução deste script, é necessária a introdução, como argumento final, de um número de segundos usado no cálculo das estatísticas referidas anteriormente.

Para além disto, é permitida a visualização da tabela de diferentes maneiras. Por defeito, a tabela está ordenada por ordem inversa da taxa de leitura dos processos, mas podemos recorrer à ordenação por quantidade de bytes escritos (com argumento -w). Também é possível a visualização inversa das ordens anteriores (com argumento -r).

Acrescentando, também deverá ser possível o filtro de processos num intervalo de tempo específico (com parâmetros inicial\_Date “-s” e final\_Date “-e”), de processos por utilizador (com parâmetro -u) e de processos com ID’s dentro de uma determinada gama (com parâmetros min\_pids “-m” e Max\_pids “-M”).

Ao longo do restante relatório, explicaremos melhor os métodos usados para a concretização do esperado script.

# **Metodologia**

De agora em diante, iremos explicar os métodos utilizados para a elaboração deste script, bem como a resolução de erros que surgiram na elaboração do trabalho.

O trabalho foi dividido em 3 partes: a primeira focou-se na declaração das variáveis a serem utilizadas e na validação de argumentos/parâmetros escritos no terminal por parte do utilizador, a segunda focou-se na pesquisa e ordenação dos dados dos processos pretendidos e a terceira focou-se na apresentação desses dados numa tabela formatada e tratamento de erros e exceções.

## **Primeira Parte**

### 1)Argumentos de entrada

Efetuando a execução do script, podemos inserir diferentes parâmetros de entrada que por um lado permitem ordenar a tabela de diferentes maneiras, assim como filtrar os processos através das opções enumeradas na introdução.

Deste modo, criámos uma função ( inputs() ) onde usámos um ciclo *while* (*while getops* -> método para processar os parâmetros colocados pelo utilizador no terminal). Os parâmetros de entrada aceites são os que estão contidos nas aspas, mas os que são sucedidos por dois pontos (:) significam que necessitam de ser seguidos por um argumento. A variável *opt* representa a opção que está a ser analisada pelo getops.

Foram criadas duas funções auxiliares ( numInt() e validarNumProcessos() ), uma verifica se o número introduzido é inteiro positivo se assim houver necessidade e a outra valida o número de processos pretendidos.

Uma imagem com texto

Descrição gerada automaticamente

Fig. 1 | ciclo while para processamento dos parâmetros colocados pelo utilizador

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto, ecrã, captura de ecrã

Descrição gerada automaticamente

Fig. 3 | Função auxiliar validarNumProcessos()

Fig.2 | Função auxiliar numInt()

De seguida, usámos um case para desempenhar diferentes funções consoante o parâmetro que está a ser avaliado (variável opt):

* -s (data inicial dos processos) -> guardamos o argumento deste na variável date e caso se verifique que é válido, a data é guardada em inicial\_Date;
* -e (data final dos processos) -> guardamos o argumento deste na variável date e caso se verifique que é válido, a data é guardada em final\_Date;
* -c (nome do processo) -> guardamos o argumento deste na variável comm;
* -u (nome do utilizador) -> guardamos o argumento deste na variável user;
* -p (número de processos a serem tabelados) -> guardamos o argumento deste na variável numero\_processos, após verificar que é um valor válido;
* -m (gama mínima de PID’s dos processos a serem tabelados) -> guardamos o argumento deste na variável min\_pids, após verificar que é um valor válido;
* -M (gama máxima de PID’s dos processos a serem tabelados) -> guardamos o argumento deste na variável Max\_pids, após verificar que é um valor válido;
* -w (ordenação da tabela) -> parâmetro que não necessita de argumento, que promove a ordenação dos processos na tabela por ordem decrescente dos seus valores escritos de bytes;
* -r (ordenação da tabela) -> parâmetro que não necessita de argumento, tomando a variável reverse o valor “1”, que promove a ordenação inversa dos processos na tabela consoante a opção de ordenação escolhida anteriormente.

Introduziu-se o comando *“shift $((OPTIND - 1))”,* pois queremos excluiros parâmetros que já goram processados pelo *getopts.*

### 2)Inicialização de variáveis

Uma imagem com texto

Descrição gerada automaticamente Primeiramente, também foram inicializadas variáveis caso certos parâmetros de entrada não fossem selecionados.

Fig. 4 | Variáveis pré-inicializadas

Por defeito, se não for atribuído o parâmetro -c, a variável *comm* terá o valor “.\*” que permitirá tabelar todos os processos com os mais variados nomes. Da mesma forma, se o parâmetro -u não for atribuído, a variável *user* terá o valor “\*” permitindo tabelar os processos ativos de todos os utilizadores.

Existem outras variáveis que também têm um valor já predefinido, como *min\_pids* e *Max\_pids* com valor “*null*” caso não seja pretendido um intervalo de PID’s, inicial\_Date com valor “0”, pois pretende-se os dados dos processos desde sempre e *final\_Date* com o valor da data do momento, pois apenas é possível processos até ao momento de execução do script.

Por último, a variável *numero\_processos*, é inicializada com o valor “null”, mas é atualizada após a busca dos processos pretendidos.

## **Segunda Parte**

### 1)CD /PROC

Para a busca e cálculo das estatísticas a serem tabeladas, teremos de aceder ao diretório ***proc*** do Linux, onde se encontram inúmeros ficheiros relativos ao funcionamento do sistema.

### 2)listas para armazenamento de dados

Para armazenar os diferentes tipos de dados, foram declaradas quatro listas no início do script, sendo estas “*processID*”, “*infoProcess*”, “*allRchar*” e “*allWchar*”, e, respetivamente, servem para guardar os números identificadores dos processos (PID), guardar todos os dados dos processos considerados válidos, guardar o número de carateres lidos da memória pelos diferentes processos antes do comando *sleep* e guardar o número de carateres escritos no disco pelos diferentes processos antes do comando *sleep*.

### 3)processos válidos

Nesta etapa, recorremos a um *for loop*, que irá percorrer todos os ficheiros em **/proc** e operar apenas sobre os diretórios que sejam números (representam os identificadores de processos).

Uma imagem com texto

Descrição gerada automaticamente

Fig.5 | for loop de validação e filtragem de processos

Passemos agora para o corpo deste for loop:

1. O primeiro *if* irá verificar se os ficheiros *status*, *io* e *comm* do processo, que o *for loop* está a iterar, existem;
2. Se verificar a condição anterior, o seguinte *if* irá verificar se temos permissões de leitura dos mesmos;
3. Após a verificação destes, o seguinte *if* irá verificar se as informações *rchar* e *wchar* estão presentes no ficheiro *io;*
4. Caso as condições anteriores se verifiquem irão ocorrer as seguintes situações:

* A variável *pComm* irá tomar o nome do processo presente no ficheiro *comm*;
* A variável *pUser* irá tomar o utilizador a que o processo está ativo;
* A variável *startDate* irá tomar a data em que o processo foi iniciado;
* A variável *data\_seg* irá tomar o valor da *startDate*, mas transformada em segundos apenas;

1. Após todo este processo, existem vários *if’s* com o intuito de filtrar os processos a guardar, de acordo com os parâmetros passados no terminal por parte do utilizador caso o tenha feito (explicado anteriormente);
2. Por fim, apresenta-se outro *if*, para assegurar que não passam processos repetidos.

Por consequência disto, o passo seguinte é invocar a função validarNumProcessos(), que verificará se o utilizador pede mais processos (parâmetro -p) do que os que foram armazenadas através do ciclo já detalhado. Caso o utilizador não tenha atribuído o parâmetro -p, a variável *numero\_processos* tomará o valor de processos captados.

### 4)Procura e armazenamento das informações de cada processo

#### 4.1)rchars e wchars iniciais

Num ciclo for, iremos guardar os valores de *rchar* e *wchar*, respetivamente nas variáveis *rvalue* e *wvalue* de cada processo já captado (*array processID*). As variáveis *rchar* e *wchar* vão armazenar unicamente os carateres numéricos. De seguida, *rchar* e *wchar* serão armazenados, respetivamente, em *allRchar* e *allWchar*.

A etapa seguinte é executar o comando *sleep* com a variável *sT* que para o processo pela quantidade de segundos introduzidos (último argumento passado no terminal

#### 4.2) rchars e wchars finais

Após o comando *sleep* terminar, voltaremos a um *for loop*. Dentro deste, iremos buscar os mais recentes valores de *rchar* e *wchar* que serão guardados nas variáveis *rvalue2* e *wvalue2*, respetivamente. Também colocaremos os valores de *rchar* e *wchar* lidos anteriormente nas variáveis *rvalue* e *wvalue*.

#### 4.3)Cálculo e armazenamento das informações

Para calcular a taxa de leitura/escrita subtrairemos *rchar2*/*wchar2* mais recente pelo *rchar*/*wchar* mais antigo, dividindo-se o resultado pelo valor de *sT* usando uma calculadora precisa (*bc*).

O próximo passo, consiste em guardar o nome do processo (*comm*), substituindo todos os espaços em branco por um sublinhado, a data (*date*) do processo e, por último, o utilizador (*user*) do processo.

Após termos todas as informações pretendidas guardadas em variáveis, iremos adicioná-las ao *array* “*infoProcess*”.

Concluindo, em cada iteração do *for loop* iremos recolher os dados de cada processo válido adicionando-os ao *array* “*infoProcess*”.

Uma imagem com texto

Descrição gerada automaticamente



Fig. 7 | Valores de rchar e wchar antes de sleep

Fig. 6 | Cálculo e armazenamento de informações

## **Terceira parte**

### 1)print da tabela

Por fim, como já temos guardados todos os processos válidos com as respetivas informações, consoante os argumentos de entrada inseridos no terminal aquando da execução do script, podemos imprimir os mesmos.

Teremos diferentes maneiras de ordenar a tabela:

* Ordem inversa da taxa de leitura (por defeito)
* Ordem da taxa de leitura -> parâmetro -r
* Ordem decrescente de bytes escritos -> parâmetro -w
* Ordem crescente de bytes escritos -> parâmetros -w e -r

Cada um destes processos segue um comando “*head -n ${numero\_processos}*” que apresenta um número limitado de processos caso o utilizador tenha usado o parâmetro -p.

Uma imagem com texto

Descrição gerada automaticamente

Fig. 8 | Função print() com os métodos de sort

A função *printInfo()* utlizada na função *print()* principal, destina-se, apenas, a formatar todos os dados de todos os processos que vão ser tabelados, tendo estes já sido filtrados antes de acordo com os parâmetros introduzidos.

Fig. 9 | Função printInfo() que complementa a função print()

### 2)Erros e exceções

A última etapa no desenvolvimento do script foi o tratamento de erros/exceções que poderiam ocorrer aquando da execução do mesmo.

Alguns destes erros/exceções são:

* Comandos diferentes dos disponíveis;
* Inserir datas de início e fim válidas;
* Data de início inferior à data final;
* Valor mínimo de número de PID’s inferior ao valor máximo;
* Alguns números necessitarem de ser inteiros positivos;

Para cada erro/exceção que venha a ocorrer, aparecerá uma mensagem de erro com a respetiva explicação e sairá do programa.

Termina assim o desenvolvimento do script que nos foi proposto.

# **Validação da solução**

Realizámos alguns testes de código para verificar a existência de possíveis falhas. Os resultados foram os seguintes:

## Erros e avisos

Fig. 1| Erro por não conter o argumento obrigatório (segundos)



Fig. 8 | Erro por número de segundos introduzidos inválido

Fig. 7 | Erro por número de segundos introduzidos inválido

Fig. 6 | Erro por opção introduzida inválida

Fig. 5 | Erro de data introduzida inválida

Fig. 4 | Erro por número de processos não ser válido

Fig. 3 | Erro pela data final ser menor que a data inicial

Fig. 2 | Erro por não ter sleep time, mas com mais parâmetros inseridos

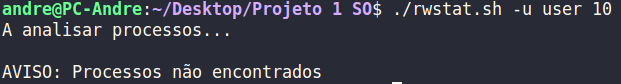


Fig. 10 | Aviso de processos não encontrados, pois não existe o usuário "user"

Fig. 9 | Aviso de processos não encontrados, pois foram pedidos 0 processos

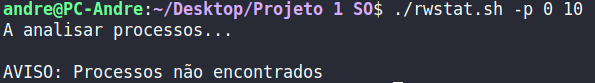


Fig. 11 | Erro pela margem superior da gama de PID's ser menor que a margem inferior

## TABELAS DE PROCESSOS

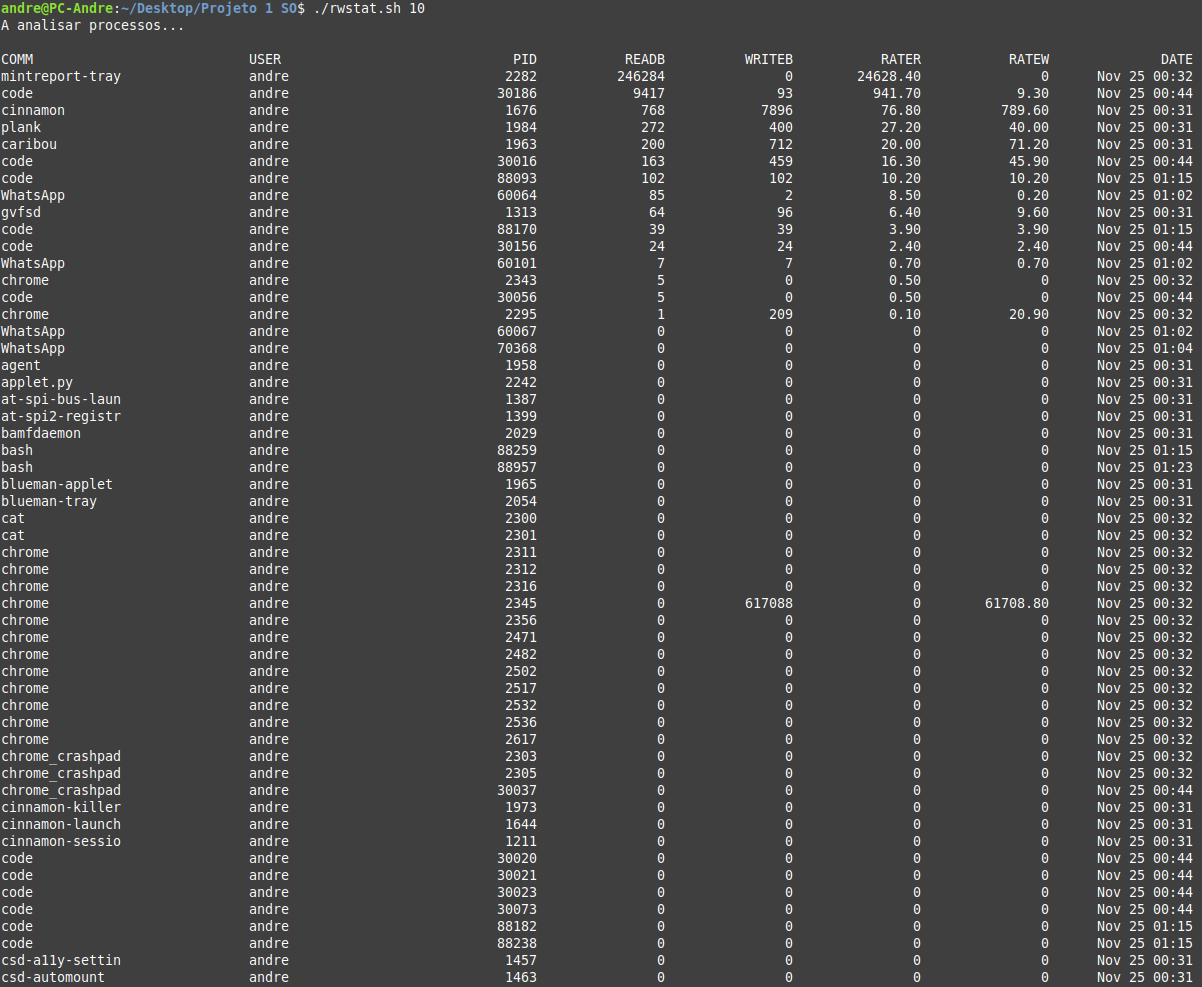


Fig. 12 |Todos os processos captados com sleep time 10 (não foi possível tirar print a todos, pois não cabe)

Fig. 13 | Processos com PID's entre 1500 e 2000 e sleep time 10

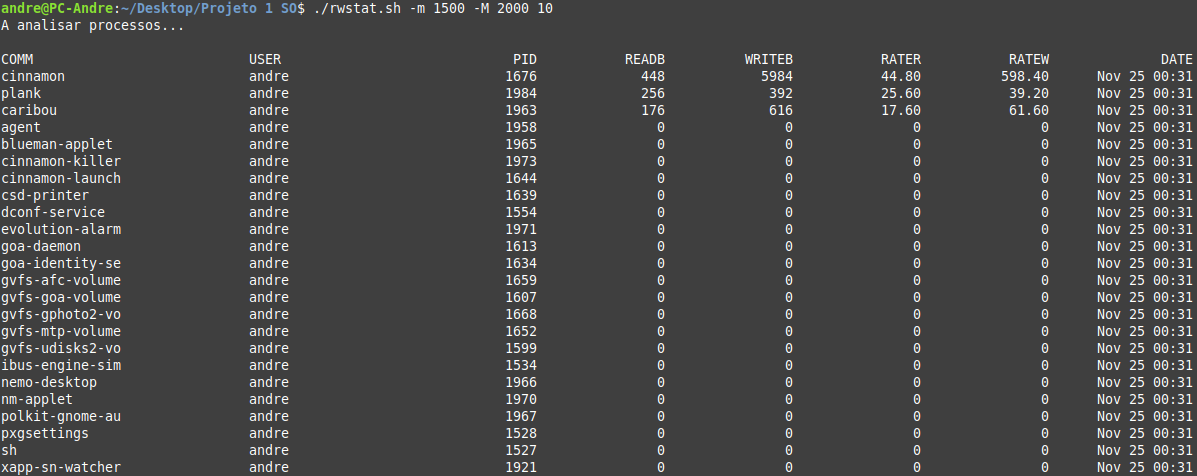


Fig. 13 | Processos com PID's entre 1500 e 2000 e sleep time 10

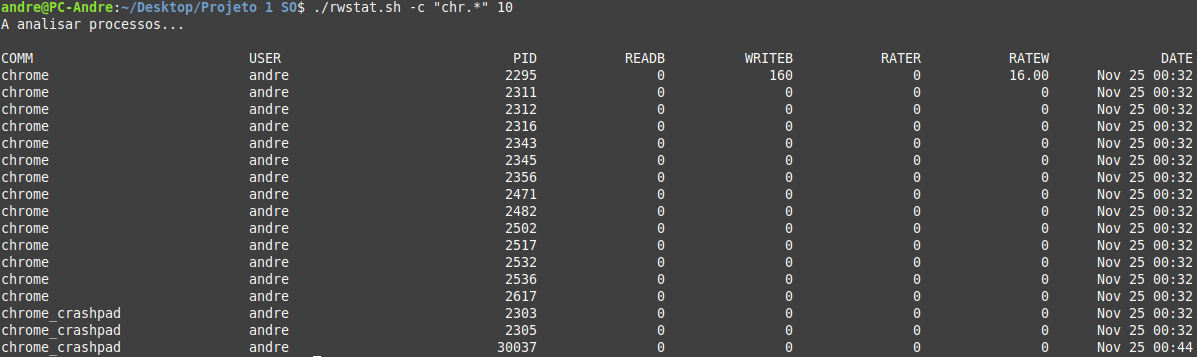


Fig. 14 | Processos em que os seus nomes contêm "chr" e sleep time 10

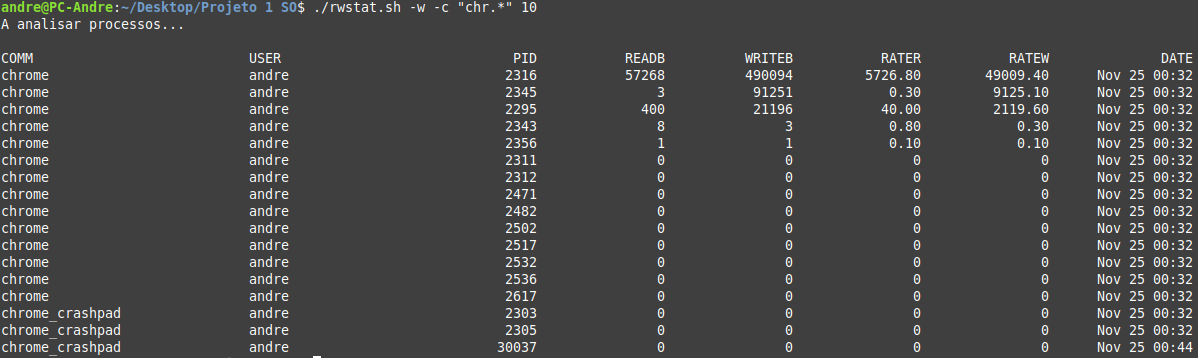


Fig. 15 | Processos em que os seus nomes contêm "chr", estão ordenados por ordem decrescente dos bytes escritos e sleep time 10

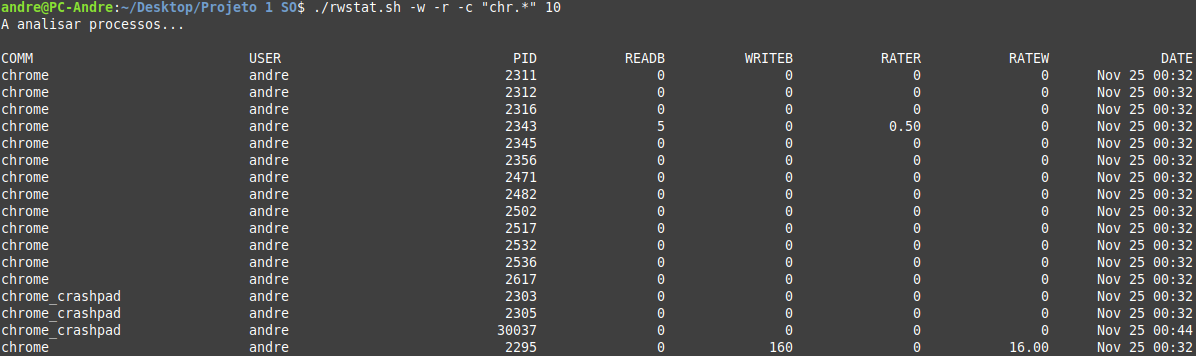
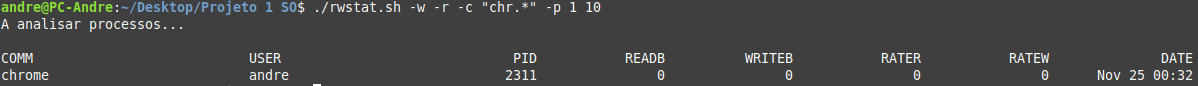


Fig. 17 | Um processo que o seu nome contém "chr", ordenado por ordem crescente dos bytes escritos e sleep time 10

Fig. 16 | Processos em que os seus nomes contêm "chr", estão ordenados por ordem crescente dos bytes escritos e sleep time 10

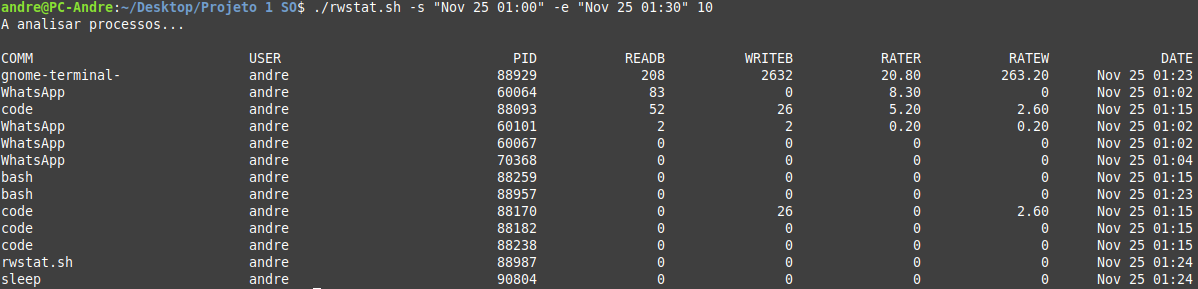


Fig. 18| Processos com começo a 25 de Novembro às 01:00 e com fim a 25 e Novembro às 01:30 e sleep time 10

# **Conclusão**

Concluindo, a resolução deste problema foi um sucesso, pois obtivemos os resultados expectáveis.

Algumas coisas foram mais fáceis do que outras, foi necessária muita pesquisa, mas no fim adquirimos vários conhecemos bastante importantes e estamos orgulhosos do resultado.