universidade de aveiro

Sistemas Operativos

Estatísticas de processos em bash

2020/2021

Diana Elisabete Siso Oliveira, nº 98607, P5
Marta Sofia Azevedo Fradique, nº 98629, P5

Índice

1. Sumário

Com o desenvolvimento do presente projeto, visamos aprimorar as nossas capacidades de programação, aumentando o conhecimento sobre o modo de funcionamento da *bash* através da realização de um *shell script* em *bash*.

O resultado final do *script* deverá apresentar no terminal estatísticas sobre os dados referentes a cada processo que se encontra em atividade no momento em que o *script* é executado, mais concretamente estatísticas sobre a quantidade de memória total, a quantidade de memória física ocupada, o número total de *bytes* de I/O que um processo escreveu/leu e a respetiva taxa de escrita/leitura num determinado intervalo de tempo.

Para a elaboração do *script*, aplicamos os conhecimentos adquiridos ao longo do semestre em conjunto com as informações aprendidas individualmente.

2. Validação de argumentos

Para que o nosso *script* possa ser executado com sucesso, é necessário verificar se todos os parâmetros passados como argumentos apresentam uma estrutura correta, de forma a que o programa não evidencie qualquer erro na sua execução devido aos elementos escritos no terminal.

O tratamento dos argumentos é, então, dividido em três partes: validação do **número de argumentos**, validação do argumento **tempo** e validação dos argumentos de **opções de filtragem** de conteúdo.

2.1 Validação do número de argumentos

O nosso *script* terá que possuir obrigatoriamente pelo menos um argumento, sendo esse correspondente ao tempo - iremos abordá-lo no próximo tópico -, pelo que as primeiras linhas de código do nosso projeto irão realizar a verificação do número de argumentos introduzidos no terminal. Caso esse número seja inferior a 1 (um), é impressa uma mensagem, indicando que é necessário ser introduzido pelo menos 1 (um) argumento, e o programa é terminado.

```
if [ $# -lt 1 ]; then
      echo "Número de argumentos inválido. Passe pelo menos 1 argumento.";
      exit;
fi
```

2.2 Validação do argumento tempo

Em relação ao argumento tempo, podemos assumir que este é passado sempre como último parâmetro da lista de argumentos, independentemente de quantos parâmetros são passados. Assim sendo, armazenamos o seu valor na variável $LASTARG.

```
LASTARG="${@: -1}"
```

A validação do conteúdo do $LASTARG consiste na verificação se foi passado um argumento que contém apenas números positivos; caso este não passe na verificação, é impressa no terminal uma mensagem indicando que é necessário que o argumento seja um número inteiro positivo.

```
if ! [[ $LASTARG =~ ^[0-9]+$ ]] || [[ $LASTARG -eq 0 ]]; then
    echo "Argumento inválido. Passe um número inteiro positivo como argumento.";
    exit;
fi
```

Observe que no código utilizado, a primeira parte da condição *if* filtra a passagem de argumentos apenas numéricos, enquanto que a segunda parte filtra a passagem de valores positivos, de modo a que só é possível passar números positivos como último argumento.

2.3 Validação dos argumentos de opções de filtragem

A validação dos argumentos de opções de filtragem de conteúdo é dividida em duas etapas: validação geral, onde averiguamos se uma opção de ordenação é inserida mais do que uma vez, e validação aprofundada, onde analisamos caso a caso detalhadamente dentro de um *switch case*.

Validação geral:

Uma opção de ordenação não pode ser passada como argumento mais do que uma vez, como já foi referido, pelo que, antes de tentarmos obter qualquer informação para a formação da tabela e depois de validarmos o parâmetro referente ao intervalo de tempo de espera, iniciamos 10 (dez) contadores, um para opção de filtragem. Aproveitamos para inicializar um contador geral responsável por incrementar 1 (um) cada vez que se depara com as opções -m, -t, -d, -r, -w e 2 (dois) cada vez que se depara com as opções -c, -e, -s, -p e -u, pois estas carecem de um parâmetro associado. Este contador servirá para validar se todos os argumentos passados serão utilizados (repare que ao correr o programa da seguinte maneira ./procstat.sh -r 2 2 2 2 2 2 2 2 2 2, apenas o último 2 (dois) será utilizado no código). O contador será inicializado contendo o valor 1 (um), simbolizando o argumento referente ao intervalo de tempo entre leituras do *rchar* e do *wchar*.

```
opP=0; opR=0; opM=0; opT=0; opW=0; opD=0; opE=0; opS=0; opU=0; opC=0
countArgs=1
```

De seguida, dentro de um ciclo *for* que percorre todos os elementos passados como argumentos, incrementamos o contador que corresponde à opção analisada no momento, recorrendo a condições *if*, e o contador referente aos argumentos utilizados ao longo do código.

```
for element in "${@}";do
    if [[ $element == "-m" ]]; then
       opM=$((opM+1))
       countArgs=$(($countArgs+1))
    fi
    if [[ $element == "-t" ]]; then
       opT=$((opT+1))
       countArgs=$(($countArgs+1))
    fi
    if [[ $element == "-d" ]]; then
       opD=$((opD+1))
       countArgs=$(($countArgs+1))
    fi
    if [[ $element == "-w" ]]; then
       opW=$((opW+1))
       countArgs=$(($countArgs+1))
    fi
    if [[ $element == "-r" ]]; then
```

```
            opR=$((opR+1))
            countArgs=$(($countArgs+1))
        fi
        if [[ $element == "-p" ]]; then
            opP=$((opP+1))
            countArgs=$(($countArgs+2))
        fi
        if [[ $element == "-c" ]]; then
            opC=$((opC+1))
            countArgs=$(($countArgs+2))
        fi
        if [[ $element == "-u" ]]; then
            opU=$((opU+1))
            countArgs=$(($countArgs+2))
        fi
        if [[ $element == "-e" ]]; then
            opE=$((opE+1))
            countArgs=$(($countArgs+2))
        fi
        if [[ $element == "-s" ]]; then
            opS=$((opS+1))
            countArgs=$(($countArgs+2))
        fi
    done
```

Para que passemos na validação com sucesso, é necessário que nenhum contador associado às opções de filtragem tenha sido incrementado mais do que 1 (uma) vez e que o valor final da variável $countArgs, o contador geral, seja igual ao número de argumentos passados. Assim sendo, através de condições *if*, realizamos comparações: entre os valores associados a cada contador de opções e o valor 1 (um) e entre o valor final do contador de argumentos utilizados e o número de parâmetros passados. Se o valor de cada contador de opções for superior a 1 (um) ou nem todos os elementos passados como argumentos sejam úteis na execução do código, ou, ainda, se estiver em falta parâmetros associados a determinadas opções de filtragem, uma mensagem será exibida no terminal, indicando que não é permitido inserir a mesma opção de ordenação múltiplas vezes ou indicando que houve passagem de argumentos extras ou faltam argumentos, e o programa irá terminar.

```
    if [[ $opS -gt 1 ]] || [[ $opE -gt 1 ]] || [[ $opD -gt 1 ]] || [[ $opC -gt 1 ]] ||
[[ $opU -gt 1 ]] || [[ $opT -gt 1 ]] || [[ $opM -gt 1 ]] || [[ $opP -gt 1 ]] || [[ $opR
-gt 1 ]] || [[ $opW -gt 1 ]]; then
          echo "Argumentos inválidos. Não introduza a mesma opção de filtragem de
conteúdo mais do que uma vez."
        exit
    fi
    if [[ $countArgs -ne $# ]]; then
         echo "Argumentos inválidos. Passou argumentos que não serão usados ou faltam
argumentos!"
        exit
    fi
```

Validação aprofundada:

A validação aprofundada será realizada dentro de um *switch case* onde é aplicado o comando *getopts*.

- Parâmetro -m:

  Para além de não poder aparecer mais do que uma vez quando é passado como argumento, o parâmetro -m não pode ser acompanhado das opções -t, -d e -w. Para analisar os casos em que tal acontece, iniciamos um ciclo *for* que percorre todos os elementos passados como argumentos e realizamos uma condição *if*. Se durante o ciclo *for* forem encontradas as opções mencionadas anteriormente, é impressa uma mensagem no terminal e o programa termina.

```
count=1
for i in "$@"; do
      if [[ "$i" == "-t" ]] || [[ "$i" == "-d" ]] || [[ "$i" == "-w" ]];
      then
            echo "Argumentos inválidos! Não pode passar a opção -m com as
opções -t, -d ou -w."
             exit
      fi
      ...
      count=$(($count+1))
done
```

- Parâmetro -t:

  Para além de não poder aparecer mais do que uma vez quando é passado como argumento, o parâmetro -t não pode ser acompanhado das opções -m, -d e -w. Para analisar os casos em que tal acontece, iniciamos um ciclo *for* que percorre todos os elementos passados como argumentos e realizamos uma condição *if*. Se durante o ciclo *for* forem encontradas as opções mencionadas anteriormente, é impressa uma mensagem no terminal e o programa termina.

```
count=1
for i in "$@"; do
      if [[ "$i" == "-m" ]] || [[ "$i" == "-d" ]] || [[ "$i" == "-w" ]];
      then
            echo "Argumentos inválidos! Não pode passar a opção -t com as
opções -m, -d ou -w."
             exit
      fi
      ...
      count=$(($count+1))
done
```

- Parâmetro -d:

  Para além de não poder aparecer mais do que uma vez quando é passado como argumento, o parâmetro -d não pode ser acompanhado das opções -t, -m e -w. Para analisar os casos em que tal acontece, iniciamos um ciclo *for* que percorre todos os elementos passados como argumentos e realizamos uma condição *if*. Se durante o ciclo *for* forem encontradas as opções mencionadas anteriormente, é impressa uma mensagem no terminal e o programa termina.

```
count=1
for i in "$@"; do
      if [[ "$i" == "-t" ]] || [[ "$i" == "-m" ]] || [[ "$i" == "-w" ]];
      then
            echo "Argumentos inválidos! Não pode passar a opção -d com as
opções -t, -m ou -w."
             exit
      fi
      ...
      count=$(($count+1))
done
```

- Parâmetro -w:

  Para além de não poder aparecer mais do que uma vez quando é passado como argumento, o parâmetro -w não pode ser acompanhado das opções -t, -d e -m. Para analisar os casos em que tal acontece, iniciamos um ciclo *for* que percorre todos os elementos passados como argumentos e realizamos uma condição *if*. Se durante o ciclo *for* forem encontradas as opções mencionadas anteriormente, é impressa uma mensagem no terminal e o programa termina.

```
count=1
for i in "$@"; do
      if [[ "$i" == "-t" ]] || [[ "$i" == "-d" ]] || [[ "$i" == "-m" ]];
      then
            echo "Argumentos inválidos! Não pode passar a opção -w com as
opções -t, -d ou -m."
             exit
      fi
      ...
      count=$(($count+1))
done
```

- Parâmetro -r:

  A validação do parâmetro -r é feita com o objetivo de, caso uma das opções de ordenação -m, -t -d e -w sejam passadas como argumentos, verificar se o *index* que esta ocupa corresponde ao *index* anterior ao que o parâmetro -r ocupa. Isto porque, caso o parâmetro -r fosse passado antes, o nosso programa

assumiria que era necessário ordenar a tabela final por ordem alfabética, sendo essa ordenação desfeita posteriormente pela ordenação correspondente à próxima opção passada (-m/-t/-d/-w). Para evitar essa incompatibilidade entre opções, se o objetivo for ordenar por ordem crescente de memória, por exemplo, terão que ser passados os argumentos pela ordem -m -r em vez de -r -m.

Dividimos então a validação da opção -r em duas secções: quando são passados apenas 3 (três) argumentos e quando são passados mais do que 3 (três) argumentos. Observe que se forem passados 2 (dois) argumentos, como o último representa o intervalo de tempo entre as leituras de *rchar* e *wchar*, o primeiro parâmetro será obrigatoriamente a opção -r, pelo que não é necessário verificar a validação.

Validação quando são passados 3 (três) argumentos:

Tendo em conta que as opções de ordenação -c, -p, -e, -s e -u necessitam de um parâmetro seguinte, não será possível conjugá-las com a opção -r e o número total de argumentos ser igual a 3 (três), pelo que as únicas opções possíveis serão -m, -t, -d e -w. Em relação a essas opções, é necessário apenas verificar, como foi explicado anteriormente, se as mesmas se encontram no *index* anterior ao correspondente ao parâmetro -r; neste caso, se as mesmas são o primeiro argumento a ser passado para a execução do código. Realizamos essas verificação através de uma condição *if*: se o parâmetro -r corresponder ao primeiro argumento que é passado, isso significa que um dos parâmetros -m, -t, -d ou -w é passado como segundo argumento, o que resulta numa mensagem impressa no terminal e no término do programa.

```
elif [[  $# -eq 3 ]]; then
      if [[ "$1" == "-r" ]]; then
              echo "Argumentos inválidos! Verifique se passou os argumentos
-m, -t, -d ou -w antes do -r ou se passou argumentos válidos."
             exit
      fi
else
```

Validação quando são passados mais do que 3 (três) argumentos:

A validação para um número de argumentos maior do que 3 (três) é um pouco mais complexa; consiste na realização de dois ciclos *for*, o primeiro para a obtenção do *index* correspondente ao parâmetro -r, e o segundo para análise com os restantes argumentos passados. A opção de ordenação -r não apresenta qualquer incompatibilidade com as opções -c, -s, -e, -p e -u (quando algumas delas é passada em conjunto com o -r, os resultados finais irão ser apresentados em ordem alfabética inversa), pelo que, mais uma vez, interessa-nos saber se foi passado um

dos 4 (quatro) parâmetros -m, -t, -d e -w e, em caso positivo, o *index* desse parâmetro. Se o *index* correspondente a uma das 4 (quatro) opções mencionadas não condizer com o *index* anterior ao do argumento -r, será exibida uma mensagem de erro no terminal.

```
indexR=1
for i in "${@}"; do
      indexO=1
      for k in "${@}"; do
            if [[ "$i" == "-r" ]]; then
                  if [[ "$k" == "-m" ]] || [[ "$k" == "-t" ]] || [[ "$k"
== "-d" ]] || [[ "$k" == "-w" ]]; then
                        if [[ $indexO -gt $indexR ]]; then
                              echo "Argumentos inválidos! Verifique se
passou os argumentos -m, -t, -d ou -w antes do -r."
                              exit
                        else
                              DIF=`expr $indexR - $indexO`
                              if [[ $DIF -ne 1 ]]; then
                                    echo "Argumentos inválidos! Verifique
se as opções -m/-t/-d/-w estão exatamente antes do -r."
                                     exit
                              else
                                    break;
                              fi
                        fi
                  ...
```

- Parâmetro -p:

  A validação da opção -p passa apenas por averiguar se o parâmetro associado é um número inteiro positivo, sendo, portanto, semelhante à validação do intervalo de tempo passado como último argumento.

```
if ! [[ $OPTARG  =~ ^[0-9]+$ ]] || [[ $OPTARG -lt 1 ]]; then
       echo "Argumento inválido. Ao parâmetro -p deverá estar associado um
número inteiro positivo."
      exit
fi
```

- Parâmetro -s e -e:

  A validação realizada para o parâmetro -s é semelhante à validação realizada para o parâmetro -e, consistindo na verificação se o próximo argumento corresponde a uma data. Assumindo que a estrutura de uma data é interpretada, para este programa, como "mês dia horário", fazemos a validação de cada um dos três elementos. Inicializamos, então, um ciclo *for* que irá percorrer todos os elementos da data - mês, dia e horário - e um contador, começado em 0 (zero) que incrementa a cada elemento.

Assim sendo, quando o contador tiver valor 0 (zero) analisamos se o mês introduzido possui uma sintaxe correta e, se possuir, atribuímos já um valor numérico entre 1-12 (1 corresponde a janeiro, 2 corresponde a fevereiro...); quando o contador tiver valor 1 (um) analisamos se o dia introduzido, tendo em conta o mês previamente analisado, possui um valor válido; quando o contador tiver valor 2 (dois) analisamos se o horário introduzido possui os parâmetros horas e minutos, no mínimo, e se estes se encontram entre intervalos de tempo reais.

```
count=0
for element in $OPTARG; do
       if [[ $count -eq 0 ]]; then
            if [[ "$element" == "Dez" ]] || [[ "$element" == "Dec" ]]; then
                  mesArg=12
            elif [[ "$element" == "Nov" ]]; then
                  mesArg=11
            elif [[ "$element" == "Oct" ]] || [[ "$element" == "Out" ]];
then
                  mesArg=10
            elif [[ "$element" == "Sep" ]] || [[ "$element" == "Set" ]];
then
                  mesArg=9
            elif [[ "$element" == "Ago" ]] || [[ "$element" == "Aug" ]];
then
                   mesArg=8
            elif [[ "$element" == "Jul" ]]; then
                   mesArg=7
            elif [[ "$element" == "Jun" ]]; then
                  mesArg=6
            elif [[ "$element" == "May" ]] || [[ "$element" == "Mai" ]];
then
                  mesArg=5
            elif [[ "$element" == "Apr" ]] || [[ "$element" == "Abr" ]];
then
                  mesArg=4
            elif [[ "$element" == "Mar" ]]; then
                  mesArg=3
            elif [[ "$element" == "Feb" ]] || [[ "$element" == "Fev" ]];
then
                  mesArg=2
            elif [[ "$element" == "Jan" ]]; then
                  mesArg=1
            else
                  echo "Primeiro argumento inválido: passe um mês como
primeiro argumento."
                  exit
            fi
      fi
...
```

Note que o programa aceita que sejam introduzidos meses em inglês ou em português e que, quando a sintaxe do mês não está correta, é emitida uma mensagem no terminal avisando o utilizador que o argumento referente ao mês não é adequado.

```
...
      if [[ $count -eq 1 ]]; then
            diaArg=$element;
            if ! [[ $element =~ ^[0-9]+$ ]]; then
                  echo "Segundo argumento inválido: passe um número
inteiro."
                  exit
            fi
            if [[ $mesArg -eq 12 ]] || [[ $mesArg -eq 10 ]] ||
               [[ $mesArg -eq 8 ]] || [[ $mesArg -eq 7 ]]  ||
               [[ $mesArg -eq 5 ]] || [[ $mesArg -eq 3 ]]  ||
               [[ $mesArg -eq 1 ]]; then
               #meses com 31 dias
                  if [[ $element -lt 1 ]] || [[ $element -gt 31 ]]; then
                        echo "Segundo argumento inválido: passe um número
compreendido entre 1 e 31."
                        exit
                  fi
            elif [[ $mesArg -eq 11 ]] || [[ $mesArg -eq 9 ]] ||
                 [[ $mesArg -eq 6 ]] || [[ $mesArg -eq 4 ]]; then
                 #meses com 31 dias
                  if [[ $element -lt 1 ]] || [[ $element -gt 30 ]]; then
                        echo "Segundo argumento inválido: passe um número
compreendido entre 1 e 30."
                        exit
                  fi
            else
            #fevereiro
                  if [[ $element -lt 1 ]] || [[ $element -gt 29 ]]; then
                        echo "Segundo argumento inválido: passe um número
compreendido entre 1 e 29."
                        exit
                  fi
            fi
      fi
...
```

Observe que a primeira etapa da verificação é se o parâmetro correspondente ao dia é um número e, caso não seja, o programa já termina, exibindo no terminal a mensagem de que não foi passado um número inteiro como argumento. No entanto, a condição *if* responsável por essa etapa da validação não verifica se o número introduzido é maior do que 0 (zero), pelo que sucede-se então a segunda etapa da verificação, onde averiguamos esse detalhe e se o dia é válido para o mês associado.

```
...
      if [[ $count -eq 2 ]]; then
            element="${element//:/ }"
            horaArg=$(awk '{printf $1}' <<< $element)
            minArg=$(awk '{printf $2}' <<< $element)
            segArg=$(awk '{printf $3}' <<< $element)
            if [[ ${#segArg} -eq 0 ]]; then
                  segArg=00
            fi
            if [[ ${#minArg} -eq 0 ]] || [[ ${#horaArg} -eq 0 ]]; then
                  echo "Argumento inválido. Passe o parâmetro horário com
pelo menos a informação referente às horas e minutos. Ex.: 23:43."
                  exit
            fi
            if [[ $horaArg =~ ^[0-9]+$ ]]  && [[ $minArg =~ ^[0-9]+$ ]] &&
[[ $segArg =~ ^[0-9]+$ ]]; then
                  if [[ $horaArg -gt 24 ]]; then
                        echo "Horas inválidas: passe um número inteiro
compreendido entre 0 e 24."
                        exit
                  fi
                  if [[ $minArg -gt 60 ]]; then
                        echo "Minutos inválidos: passe um número inteiro
compreendido entre 0 e 60."
                        exit
                  fi
                  if [[ $segArg -gt 60 ]]; then
                        echo "Segundos inválidos: passe um número inteiro
compreendido entre 0 e 60."
                        exit
                  fi
            else
                  echo "Terceiro argumento inválido: passe um número
inteiro."
                  exit
            fi
      fi
      count=$((count+1))
done
```

Repare que dividimos o horário em 3 (três) secções diferentes e, uma vez que julgamos não obrigatória a introdução do parâmetro segundos, se este não for passado assume-se que o seu valor é 0 (zero). No entanto, se o parâmetro correspondente às horas ou aos minutos não for passado, é mostrada uma mensagem no terminal e o programa termina de correr.

Para concluir, comparamos o valor da variável $count e caso este seja maior do que 3 (três) significa que foram passados mais parâmetros do que os necessários, sendo exibida uma mensagem no terminal e o programa termina.

```
if [[ $count -gt 3 ]]; then
          echo "Número de argumentos inválido! Passe apenas 3 argumentos, o
1° referente ao mês, o 2° referente ao dia e o 3° referente às horas."
         exit
fi
```

- Parâmetros -c e -u:

  Não carecem de qualquer validação.

3. Seleção e validação dos ficheiros

Uma vez que não é permitida a utilização do comando *sudo* para a obtenção de permissões especiais para aceder aos ficheiros que contêm os dados referentes aos processos, é necessário dividir a validação dos ficheiros utilizados em duas etapas: saber quais pastas que se encontram na diretoria */proc* serão utilizadas e, dentro das pastas selecionadas, quais é que possuem os ficheiros *status* e *io*, sendo permitida a leitura dos mesmos.

Antes de realizar qualquer etapa, iniciamos um ciclo *for* para percorrer todos os elementos presentes na diretoria */proc*, seguindo-se então a seleção e validação de cada elemento.

```
for f in /proc/*; do
      ...
done
```

3.1 Seleção das pastas referentes aos processos

Ao analisar o conteúdo da pasta *proc* reparamos que apenas será do nosso interesse aceder às pastas identificadas com apenas números - esses números representam o *ID* do processo -, pelo que realizamos uma filtragem desses processos através do comando *if*. O código restante apenas será executado para as pastas cuja diretoria possua apenas os seguintes elementos: /proc (identifica a diretoria da pasta *proc*) e 0123456789 (filtra para apenas pastas cujo nome sejam números).

```
if [[ $f =~ ^[/proc0123456789]+$ ]]; then
      ...
fi
```

3.2 Seleção dos processos que permitem leitura dos ficheiros *status* e *io*

Já com a filtragem das pastas que contêm apenas os dados de relevância para o desenvolvimento do projeto feita, procuramos realizar uma segunda filtragem de forma a selecionar apenas as pastas que contêm processos cuja leitura do respetivo ficheiro *status* e *io* seja permitida sem ser preciso recorrer ao comando *sudo*.

Para tal, dividimos a filtragem em duas etapas. A primeira etapa visa verificar se a pasta do respetivo processo tem os dois ficheiros mencionados, enquanto que a segunda etapa irá verificar, para as pastas com ambos os ficheiros, quais é que permitem a sua leitura.

```
if [ -f "$f/status" ] && [ -f "$f/io" ]
then                                    #verifica a existência dos ficheiros

            if [ -r $f/status ] && [ -r $f/io ]
            then                        #verifica a permissão de leitura
```

4. Obtenção dos dados dos ficheiros *status* e *io*

Após a realização das filtragens explicadas previamente, é hora de extrair os dados que necessitamos e armazená-los em diferentes variáveis. Todas as extrações são realizadas seguindo o mesmo método, auxiliando-nos dos comandos *cat*, *awk* e *head*.

Com o comando *cat* selecionamos qual dos dois ficheiros queremos ler:

```
cat $f/status
```
ou
```
cat $f/io
```

Com o comando *awk* filtramos qual dado queremos extrair no ficheiro. Observe o exemplo abaixo:

```
PID=$(cat $f/status | awk '/Pid:/{print $2}')
```

Neste exemplo, no ficheiro *status* do respetivo processo, será procurado o conjunto de caracteres "Pid:" e, quando encontrado, irá ser armazenada na variável $PID a palavra seguinte aos caracteres mencionados. Assumindo que a estrutura do ficheiro *status* apresenta uma linha com o conteúdo "Pid: 175", o que será armazenado na variável $PID será o número 175, correspondente ao ID do processo.

No entanto, ao longo do desenvolvimento do projeto, foi possível observar que o ficheiro *status* possui mais linhas onde o conjunto de caracteres "Pid:" aparece, linhas essas que não são do nosso interesse. Com a finalidade de evitar que informação desnecessária seja extraída, fazemos uso do comando *head* -1, que fará a extração do conteúdo para a respetiva variável parar assim que encontrar a primeira informação correspondente aos parâmetros.

Os dados referentes às 5 (cinco) primeiras colunas, com exceção da segunda coluna, da tabela - COMM, PID, MEM, RSS - são obtidos através do ficheiro *status*, enquanto que os dados referentes às colunas 6 (seis) e 7 (sete) - READB e WRITEB - são obtidos através do ficheiro *io*.

```
    READB=$(cat $f/io | awk '/read_bytes/{print $2}'| head -1)
    WRITEB=$(cat $f/io | awk '/write_bytes/{print $2}'| head -1)
    NAME=$(cat $f/status | awk '/Name:/{for (i=2; i<=NF; i++) printf "%s ", $i} END
{print ""}')
    PID=$(cat $f/status | awk '/Pid:/{print $2}' | head -1)
    MEM=$(cat $f/status | awk '/VmSize:/{print $2}' | head -1)
    RSS=$(cat $f/status | awk '/VmRss:/{print $2}' | head -1)
```

Note que a extração do nome do processo realiza-se usando um ciclo *for*, uma vez que existem processos cujo nome possui mais do que uma palavra. Posteriormente, nos casos em que o nome possui mais do que uma palavra, unimos as palavras removendo os

whitespaces e convertemos a palavra para somente letras minúsculas, com a finalidade de não prejudicar o código futuro, quando o organizarmos por ordem alfabética.

```
NAME="${NAME// /}"
NAME=$(echo "$NAME" | tr '[:upper:]' '[:lower:]')
```

De seguida, para evitar que apareçam espaços vazios na tabela exibida no terminal, caso não haja informação a ser extraída para cada uma das variáveis mencionadas, verificamos se o tamanho delas corresponde a 0 (zero), através de condições *if*, e substituímos por 0's (zeros) ou, no caso do nome do processo, por "-----".

```
if [[ ${#MEM} -eq 0 ]]; then
      MEM=0
fi
if [[ ${#RSS} -eq 0 ]]; then
       RSS=0
fi
if [[ ${#NAME} -eq 0 ]]; then
      NAME="-----"
fi
if [[ ${#WRITEB} -eq 0 ]]; then
       WRITEB=0
fi
if [[ ${#READB} -eq 0 ]]; then
       READB=0
fi
```

5. Tratamento da informação referente ao RATER e ao RATEW

Para obter os resultados da oitava e da nona colunas da tabela impressa no terminal, é necessário realizar duas leituras onde fazemos a extração dos valores referentes ao número de *bytes* lidos e escritos, sendo essas duas leituras separadas por um intervalo de tempo que corresponderá ao último argumento passado na consola.

Primeiramente iniciamos um ciclo *for* - realizado antes do ciclo *for* para as extrações explicadas no tópico 4 - que percorrerá todos os ficheiros que passem na filtragem explicada no tópico 3, com o objetivo de extrair o valor correspondente ao campo *rchar* e *wchar* do ficheiro *io* associado ao processo. A obtenção dos valores do *rchar* e do *wchar* é elaborada seguindo o mesmo padrão explicado no tópico 4, porém estes valores são armazenados inicialmente numa variável e depois transcritos para um ficheiro de texto - RATE1.txt -, tendo o ID do processo associado ao valor extraído.

```
for f in /proc/*; do
    if [[ $f =~ ^[/proc0123456789]+$ ]]; then
       if [ -f "$f/status" ] && [ -f "$f/io" ]; then
             if [ -r $f/status ] && [ -r $f/io ]; then
                   PID=$(cat $f/status | awk '/Pid:/{print $2}' | head -1)
                   RATER=$(cat $f/io | awk '/rchar/{print $2}'| head -1)
                   RATEW=$(cat $f/io | awk '/wchar/{print $2}'| head -1)
                   echo "$PID - rchar: $RATER" >> RATE1.txt
                   echo "$PID - wchar: $RATEW" >> RATE1.txt
             fi
       fi
    fi
done
```

A identificação do ID do respetivo processo ao qual os valores do *rchar* e *wchar* estão associados será importante numa etapa futura, visto que durante o intervalo de tempo até à próxima leitura é provável que alguns processos deixem de ser executados ou que novos processos passem a ser executados. Para que o programa pare de correr durante o intervalo de tempo desejado, recorremos ao comando *sleep* associado à variável $LASTARG, referida anteriormente.

```
sleep $LASTARG
```

Assim que o intervalo de tempo passar, iniciamos outro ciclo *for* - no caso, será o último ciclo *for* do programa e o mesmo que será utilizado para a extração dos dados mencionados no tópico 4 -, passando novamente pela filtragem, e obtemos os novos valores de *rchar* e *wchar*, armazenando-os nas variáveis $RATER2 e $RATEW2. Contrariamente ao que foi realizado no primeiro ciclo *for*, não será necessário transcrever os valores para um ficheiro texto, pois os mesmos serão utilizados assim que forem extraídos.

```
for f in /proc/*; do
    if [[ $f =~ ^[/proc0123456789]+$ ]]; then
       if [ -f "$f/status" ] && [ -f "$f/io" ]; then
             if [ -r $f/status ] && [ -r $f/io ]; then
                  RATER2=$(cat $f/io | awk '/rchar/{print $2}'| head -1)
                  RATEW2=$(cat $f/io | awk '/wchar/{print $2}'| head -1)
                  ...
```

Posteriormente, para o respetivo processo a ser analisado, extraímos os valores do *rchar* e do *wchar* lidos anteriormente e armazenados no ficheiro de texto - RATE1.txt. Para além dos comandos *cat*, *awk* e *head*, já explicados, fazemos uso do comando *grep* para filtrar a informação lida, utilizando a variável $PID, de forma a obter os valores apenas do processo que está a ser iterado.

```
RATER1=$(cat RATE1.txt | grep "$PID - rchar:" | awk '{print $4}'| head -1)
RATEW1=$(cat RATE1.txt | grep "$PID - wchar:" | awk '{print $4}'| head -1)
```

Já com os valores do *rchar* e do *wchar* das duas leituras armazenados em diferentes variáveis, antes de prosseguirmos para o cálculo da taxa associada, verificamos se todas as variáveis têm alguma informação guardada, ou seja, se têm comprimento maior que 0 (zero). Este passo faz-se necessário uma vez que, como referido antes, durante o intervalo de tempo entre as duas leituras, há processos que podem deixar de ser executados - esses não irão aparecer na tabela final, pois não são iterados no segundo ciclo *for* - e pode haver novos processos, os quais não terão nenhum valor de *rchar* e *wchar* lidos anteriormente, pelo que as variáveis RATER1 e RATEW1 estarão vazias. Caso exista qualquer variável sem informação, esta passa a ter valor 0 (zero).

```
if [[ ${#RATER1} -eq 0 ]]; then
      RATER1=0
fi
if [[ ${#RATER2} -eq 0 ]]; then
       RATER2=0
fi
if [[ ${#RATEW1} -eq 0 ]]; then
       RATEW1=0
fi
if [[ ${#RATEW2} -eq 0 ]]; then
       RATEW2=0
fi
```

Finalizando o tratamento da informação referente às taxas de leitura e de escrita, calculamos a diferença entre o segundo e o primeiro valores lidos, armazenando os resultados nas variáveis $RESULTR e $RESULTW. O cálculo do valor da taxa será realizado quando a tabela for construída, pelo que, por agora, já temos os dados necessários.

```
RESULTR=`expr $RATER2 - $RATER1; RESULTW=`expr $RATEW2 - $RATEW1
```

6. Formação da tabela final

O processo de formação da informação que será exibida no terminal divide-se em duas partes: formação da tabela com o máximo de dados que for possível obter em relação aos processos e filtragem de conteúdo mostrado, tendo em conta as opções passadas como argumento.

É importante mencionar que, para armazenar os dados recolhidos ao longo do código utilizamos ficheiros de texto, como já se verificou para o exemplo das taxas de leitura e de escrita. Para além do RATE1.txt, mencionado no tópico anterior, fazemos uso de mais dois ficheiros: TABELA.txt e TEMP.txt. O ficheiro TABELA.txt, num primeiro instante, contém a informação de todos os processos cujos ficheiros *status* e *io* podem ser lidos, sendo alterado posteriormente. O ficheiro TEMP.txt é um ficheiro temporário e será utilizado para passar a informação atualizada pelas diversas opções de filtragem - irá entender melhor o funcionamento a seguir.

Como tal, depois de validar o argumento correspondente ao intervalo de tempo de espera entre as duas leituras do *rchar* e do *wchar* e antes de iniciar a primeira leitura dos dois campos, aplicamos os seguintes comandos:

```
: > TABELA.txt
: > RATE1.txt
: > TEMP.txt
```

Caso os ficheiros já existam na pasta onde o utilizador está a executar o programa (essa situação ocorre se o programa já tiver sido corrido antes), as três linhas de código irão limpar o conteúdo dos três ficheiros de texto. Se os ficheiros ainda não existirem, então serão criados sem qualquer conteúdo dentro.

O próximo passo para a formação da tabela inicial será ler o valor de *rchar* e *wchar* de cada processo que permita a leitura dos ficheiros *status* e *io*. O código responsável por tal já foi explicado previamente nos tópicos 3 e 5.

```
for f in /proc/*; do
    if [[ $f =~ ^[/proc0123456789]+$ ]]; then
       if [ -f "$f/status" ] && [ -f "$f/io" ]; then
             if [ -r $f/status ] && [ -r $f/io ]; then
                   PID=$(cat $f/status | awk '/Pid:/{print $2}' | head -1)
                   RATER=$(cat $f/io | awk '/rchar/{print $2}'| head -1)
                   RATEW=$(cat $f/io | awk '/wchar/{print $2}'| head -1)
                   echo "$PID - rchar: $RATER" >> RATE1.txt
                   echo "$PID - wchar: $RATEW" >> RATE1.txt
             fi
       fi
    fi
done
```

De seguida, após aguardar o intervalo de tempo inserido pelo utilizador, prosseguimos para a segunda leitura e para a obtenção, através dos ficheiros *status* e *io*, dos valores dos campos da tabela respetivamente ao processo analisado. O código apresentado já foi explicado nos tópicos 3, 4 e 5.

```
for f in /proc/*; do
    if [[ $f =~ ^[/proc0123456789]+$ ]]; then
       if [ -f "$f/status" ] && [ -f "$f/io" ]; then
             if [ -r $f/status ] && [ -r $f/io ]; then
                   PID=$(cat $f/status | awk '/Pid:/{print $2}' | head -1)
                   RATER2=$(cat $f/io | awk '/rchar/{print $2}'| head -1)
                   RATEW2=$(cat $f/io | awk '/wchar/{print $2}'| head -1)
                   READB=$(cat $f/io | awk '/read_bytes/{print $2}'| head -1)
                   WRITEB=$(cat $f/io | awk '/write_bytes/{print $2}'| head -1)
                   NAME=$(cat $f/status | awk '/Name:/{for (i=2; i<=NF; i++)
printf "%s ", $i} END {print ""}')
                   PID=$(cat $f/status | awk '/Pid:/{print $2}' | head -1)
                   MEM=$(cat $f/status | awk '/VmSize:/{print $2}' | head -1)
                   RSS=$(cat $f/status | awk '/VmRss:/{print $2}' | head -1)
                   RATER1=$(cat RATE1.txt | grep "$PID - rchar:" | awk '{print
$4}'| head -1)
                   RATEW1=$(cat RATE1.txt | grep "$PID - wchar:" | awk '{print
$4}'| head -1)
                   if [[ ${#MEM} -eq 0 ]]; then
                         MEM=0
                   fi
                   if [[ ${#RSS} -eq 0 ]]; then
                         RSS=0
                   fi
                   if [[ ${#NAME} -eq 0 ]]; then
                         NAME="-----"
                   fi
                   NAME="${NAME// /_}"
                   NAME=$(echo "$NAME" | tr '[:upper:]' '[:lower:]')
                   if [[ ${#WRITEB} -eq 0 ]]; then
                         WRITEB=0
                   fi
                   if [[ ${#READB} -eq 0 ]]; then
                         READB=0
                   fi
                   if [[ ${#RATER1} -eq 0 ]]; then
                         RATER1=0
                   fi
                   if [[ ${#RATER2} -eq 0 ]]; then
                         RATER2=0
                   fi
                   if [[ ${#RATEW1} -eq 0 ]]; then
                         RATEW1=0
                   fi
                   if [[ ${#RATEW2} -eq 0 ]]; then
                         RATEW2=0
                   fi
```

```
RESULTR=`expr $RATER2 - $RATER1`
RESULTW=`expr $RATEW2 - $RATEW1`
```

A próxima etapa é passar a informação para o ficheiro TABELA.txt já com a formatação da tabela final. Repare que no ficheiro TABELA.txt não iremos encontrar o cabeçalho, isto porque a colocação do cabeçalho no ficheiro dificulta a ordenação da tabela consoante as opções de filtragem passadas.

Começamos por introduzir o campo COMM. Para evitar a desformatação da tabela impressa no terminal, aplicamos uma condição *if*: se o nome do processo possuir mais do que 12 (doze) caracteres, serão impressos os primeiros 10 (dez) juntamente com reticências.

```
if [[ ${#NAME} -gt 12 ]];then
      printf '%-20s' "${NAME:0:10}..." >> TABELA.txt
else
      printf '%-20s' "$NAME" >> TABELA.txt
fi
```

Seguem-se os campos do ID do processo e do *user* que o executa. A extração da informação referente ao utilizador é realizada tendo em conta o valor da variável $PID: se for nulo, então os campos USER e PID da tabela serão preenchidos com "-----"; se for diferente de nulo então obtém-se o valor do campo USER recorrendo ao comando abaixo especificado.

```
if [[ ${#PID} -eq 0 ]]; then
      PID="-----"
      printf '%-20s %10s' "-----" "$PID" >> TABELA.txt
else
      printf '%-20s %10s' "$(ps -o user= -p $PID)" "$PID" >> TABELA.txt
fi
```

Uma vez que os valores referentes aos campos MEM, RSS, READB e WRITEB já foram obtidos e analisados anteriormente, transcrevemos todos para o ficheiro ao mesmo tempo.

```
printf '%16s %15s %20s %20s' "$MEM" "$RSS" "$READB" "$WRITEB" >> TABELA.txt
```

Iremos então analisar os resultados das taxas de leitura e de escrita, tendo em conta que já estão armazenados nas variáveis $RESULTR e $RESULTW os resultados da subtração entre os valores da segunda e da primeira leitura do *rchar* e do *wchar*, ou seja, resta apenas dividir pelo último parâmetro passado como argumento. Para realizar uma divisão que devolva um número fracionário, recorremos ao comando *bc -l* e atribuímos o valor 2 (dois) ao parâmetro *scale* para que o resultado seja apresentado com 2 (duas) casas decimais.

```
printf '%21s' $(echo "scale=2; $RESULTR/$LASTARG" | bc -l ) >> TABELA.txt
printf '%21s' $(echo "scale=2; $RESULTW/$LASTARG" | bc -l ) >> TABELA.txt
```

Por último, falta apenas introduzir o conteúdo da coluna DATE, referente à data do processo. Aplicamos a lógica usada para o campo USER, visto que a data do processo é obtida usando o seu ID, portanto: se o ID for nulo, então o campo DATE da tabela será preenchidos com "-----"; se o ID for diferente de nulo então obtém-se o valor do campo recorrendo ao comando abaixo especificado.

```
    if [[ ${#PID} -eq 0 ]]; then
        printf '%20s\n' "-----" >> TABELA.txt
    else
        DATE=$(ps -o lstart= -p $PID | awk '{for (i=2; i<NF; i++) printf "%s ", $i}
END {print ""}')
        DATE="${DATE// /_}"
        DATE=${DATE::-1}
        printf '%22s\n' "$DATE" >> TABELA.txt
    fi
```

Concluído o processo de passagem dos dados obtidos até agora para o ficheiro TABELA.txt aplicamos o comando *sort* para ordenar o conteúdo da tabela por ordem alfabética, uma vez que, por defeito, se não for introduzida nenhuma opção de ordenação, será assim que a informação será apresentada no terminal.

```
    sed '1d' TABELA.txt > TEMP.txt;
    : > TABELA.txt
    sort TEMP.txt > TABELA.txt
```

Repare que não é possível realizar a ordenação das linhas por ordem alfabética sem recorrer ao ficheiro TEMP.txt. Isto acontece porque, em *bash*, não é permitido aplicar comandos de ordenação e a informação ser atualizada automaticamente no próprio ficheiro. Observe também que antes de realizarmos qualquer operação, removemos a primeira linha do ficheiro TABELA.txt com o comando *sed*, uma vez que a mesma é uma linha em branco e a sua presença pode originar erros posteriormente.

Finalizamos então a formação da tabela inicial que contém a informação referente a todos os processos que passaram na validação e seleção explicada no tópico 3 do presente relatório. Se não tiver sido passada nenhuma opção de filtragem de conteúdo, o código irá terminar após as duas linhas de código abaixo serem lidas:

```
    printf '%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %21s\n' "COMM" "USER" "PID"
"MEM" "RSS" "READB" "WRITEB" "RATER" "RATEW" "DATE"    #cabeçalho
    cat TABELA.txt     #mostrar o conteúdo do ficheiro TABELA.txt
```

Caso tenha sido passada como argumento alguma opção de filtragem, ou seja, o número de argumentos é superior a 1 (um), iremos entrar num ciclo *while* e faremos uso do comando *getopts* para ler todos os argumentos.

Vamos analisar agora como o programa irá proceder face às diferentes opções.

Opção -m:

Quando a opção -m é passada como argumento, a primeira coisa que o programa irá verificar é se foi passada a opção -r a seguir. Caso o -m esteja associado ao -r, iremos ordenar a informação da tabela por ordem crescente da coluna MEM; caso contrário, iremos ordenar a informação da tabela por ordem decrescente da coluna MEM.

Para saber se foi passado a opção -r, inicializamos as variáveis $reverse - servirá para, se for encontrado o parâmetro mencionado, assumir o valor 1 (um) - e $count - funcionará como contador para comparação. Através da variável $OPTIND, sabemos o *index* do argumento seguinte ao parâmetro -m, pelo que, se, para quando o valor de $count for igual ao valor do $OPTIND, o argumento for igual a -r, então a variável $reverse assume o valor 1 (um).

```
reverse=0
count=1
for i in "$@"; do
...
      if [[ $count -eq $OPTIND ]]; then
            if [[ "$i" == "-r" ]]; then
                  reverse=1
            fi
      fi
      count=$(($count+1))
done
```

De seguida, copiamos a informação do ficheiro TABELA.txt para o ficheiro TEMP.txt e limpamos o ficheiro TABELA.txt (se não fizermos este passo agora, teremos que fazê-lo posteriormente, de modo a que a informação final após ordenação fique no ficheiro TABELA.txt).

```
cp TABELA.txt > TEMP.txt;
: > TABELA.txt
```

Por último, verificamos o valor da variável $reverse: se for igual a 0 (zero), ordena a informação da tabela por ordem decrescente da coluna MEM; se for igual a 1 (um), ordena a informação da tabela por ordem crescente da coluna MEM. Para a ordenação, fazemos uso do comando *sort* e do comando *uniq*, este último servirá apenas para garantir que não há presença de linhas repetidas no nosso código.

```
      if [[ $reverse -eq 0 ]]; then
            awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nrk4 | uniq > TABELA.txt
```

```
	else
		awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nk4 | uniq > TABELA.txt
	fi
	;;
```

Opção -t:

Quando a opção -t é passada como argumento, a primeira coisa que o programa irá verificar é se foi passada a opção -r a seguir. Caso o -t esteja associado ao -r, iremos ordenar a informação da tabela por ordem crescente da coluna RSS; caso contrário, iremos ordenar a informação da tabela por ordem decrescente da coluna RSS.

Para saber se foi passado a opção -r, inicializamos as variáveis $reverse - servirá para, se for encontrado o parâmetro mencionado, assumir o valor 1 (um) - e $count - funcionará como contador para comparação. Através da variável $OPTIND, sabemos o *index* do argumento seguinte ao parâmetro -t, pelo que, se, para quando o valor de $count for igual ao valor do $OPTIND, o argumento for igual a -r, então a variável $reverse assume o valor 1 (um).

```
reverse=0
count=1
for i in "$@"; do
...
	if [[ $count -eq $OPTIND ]]; then
		if [[ "$i" == "-r" ]]; then
			reverse=1
		fi
	fi
	count=$(($count+1))
done
```

De seguida, copiamos a informação do ficheiro TABELA.txt para o ficheiro TEMP.txt e limpamos o ficheiro TABELA.txt (se não fizermos este passo agora, teremos que fazê-lo posteriormente, de modo a que a informação final após ordenação fique no ficheiro TABELA.txt).

```
cp TABELA.txt > TEMP.txt;
: > TABELA.txt
```

Por último, verificamos o valor da variável $reverse: se for igual a 0 (zero), ordena a informação da tabela por ordem decrescente da coluna RSS; se for igual a 1 (um), ordena a informação da tabela por ordem crescente da coluna RSS. Para a ordenação, fazemos uso do comando *sort* e do comando *uniq*, este último servirá apenas para garantir que não há presença de linhas repetidas no nosso código.

```
    if [[ $reverse -eq 0 ]]; then
        awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nrk5 | uniq > TABELA.txt

    else
        awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nk5 | uniq > TABELA.txt
    fi
    ;;
```

Opção -d:

Quando a opção -d é passada como argumento, a primeira coisa que o programa irá verificar é se foi passada a opção -r a seguir. Caso o -d esteja associado ao -r, iremos ordenar a informação da tabela por ordem crescente da coluna RATER; caso contrário, iremos ordenar a informação da tabela por ordem decrescente da coluna RATER.

Para saber se foi passado a opção -r, inicializamos as variáveis $reverse - servirá para, se for encontrado o parâmetro mencionado, assumir o valor 1 (um) - e $count - funcionará como contador para comparação. Através da variável $OPTIND, sabemos o *index* do argumento seguinte ao parâmetro -d, pelo que, se, para quando o valor de $count for igual ao valor do $OPTIND, o argumento for igual a -r, então a variável $reverse assume o valor 1 (um).

```
    reverse=0
    count=1
    for i in "$@"; do
    ...
        if [[ $count -eq $OPTIND ]]; then
            if [[ "$i" == "-r" ]]; then
                reverse=1
            fi
        fi
        count=$(($count+1))
    done
```

De seguida, copiamos a informação do ficheiro TABELA.txt para o ficheiro TEMP.txt e limpamos o ficheiro TABELA.txt (se não fizermos este passo agora, teremos que fazê-lo posteriormente, de modo a que a informação final após ordenação fique no ficheiro TABELA.txt).

```
    cp TABELA.txt > TEMP.txt;
    : > TABELA.txt
```

Por último, verificamos o valor da variável $reverse: se for igual a 0 (zero), ordena a informação da tabela por ordem decrescente da coluna RATER; se for igual a 1 (um), ordena a informação da tabela por ordem crescente da coluna RATER. Para a ordenação, fazemos

uso do comando *sort* e do comando *uniq*, este último servirá apenas para garantir que não há presença de linhas repetidas no nosso código.

```
    if [[ $reverse -eq 0 ]]; then
          awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nrk8 | uniq > TABELA.txt

    else
          awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nk8 | uniq > TABELA.txt
    fi
    ;;
```

Opção -w:

Quando a opção -w é passada como argumento, a primeira coisa que o programa irá verificar é se foi passada a opção -r a seguir. Caso o -w esteja associado ao -r, iremos ordenar a informação da tabela por ordem crescente da coluna RATEW; caso contrário, iremos ordenar a informação da tabela por ordem decrescente da coluna RATEW.

Para saber se foi passado a opção -r, inicializamos as variáveis $reverse - servirá para, se for encontrado o parâmetro mencionado, assumir o valor 1 (um) - e $count - funcionará como contador para comparação. Através da variável $OPTIND, sabemos o *index* do argumento seguinte ao parâmetro -w, pelo que, se, para quando o valor de $count for igual ao valor do $OPTIND, o argumento for igual a -r, então a variável $reverse assume o valor 1 (um).

```
    reverse=0
    count=1
    for i in "$@"; do
    ...
          if [[ $count -eq $OPTIND ]]; then
                if [[ "$i" == "-r" ]]; then
                      reverse=1
                fi
          fi
          count=$(($count+1))
    done
```

De seguida, copiamos a informação do ficheiro TABELA.txt para o ficheiro TEMP.txt e limpamos o ficheiro TABELA.txt (se não fizermos este passo agora, teremos que fazê-lo posteriormente, de modo a que a informação final após ordenação fique no ficheiro TABELA.txt).

```
    cp TABELA.txt > TEMP.txt;
    : > TABELA.txt
```

Por último, verificamos o valor da variável $reverse: se for igual a 0 (zero), ordena a informação da tabela por ordem decrescente da coluna RATEW se for igual a 1 (um), ordena a informação da tabela por ordem crescente da coluna RATEW. Para a ordenação, fazemos uso do comando *sort* e do comando *uniq*, este último servirá apenas para garantir que não há presença de linhas repetidas no nosso código.

```
    if [[ $reverse -eq 0 ]]; then
        awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nrk9 | uniq > TABELA.txt

    else
        awk '{ printf "%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %20s\n", $1,
$2, $3, $4, $5, $6, $7, $8, $9, $10 }' TEMP.txt | sort -nk9 | uniq > TABELA.txt
    fi
    ;;
```

Opção -p:

Quando a opção -p é passada como argumento, após a validação do $OPTARG associado, iremos passar para o ficheiro TEMP.txt o número de linhas correspondente ao valor do $OPTARG usando o comando *head*.

```
    head -$(($OPTARG)) TABELA.txt > TEMP.txt
    : > TABELA.txt
    cp TEMP.txt TABELA.txt
    ;;
```

Copiamos a informação novamente para o ficheiro TABELA.txt, pois é esse ficheiro que será utilizado posteriormente quando for para imprimir os dados no terminal.

Opção -u:

Quando a opção -u é passada como argumento, realizamos um ciclo *while* para a leitura de cada linha do ficheiro TEMP.txt (já terá sido executado o comando para copiar os dados do ficheiro TABELA.txt para o ficheiro mencionado). Ao ler cada linha do ficheiro, iremos extrair a informação respetiva à segunda coluna da linha - referente à coluna USER - para a variável $userT e, caso o conteúdo dessa variável seja igual ao $OPTARG, a linha analisada é copiada integralmente para o ficheiro TABELA.txt (que já se encontra vazio).

```
    cp TABELA.txt > TEMP.txt;
    : > TABELA.txt
    {
    while read line; do
        userT=$(awk '{printf $2}' <<< $line)
        if [[ "$userT" == "$OPTARG" ]]; then
            echo "$line" >> TABELA.txt
        fi
```

```
done
}<TEMP.txt
;;
```

Opção -c:

Quando a opção -c é passada como argumento, sabendo que a mesma é passada, por exemplo, no formato "*string*.*", indicando que a intenção é que seja exibido no terminal os processos *começados* pela *string*, a primeira coisa que fazemos é remover, se existir, o . do $OPTARG.

```
OPTARG="${OPTARG//./}"
cp TABELA.txt > TEMP.txt;
: > TABELA.txt
```

Ficamos então com um $OPTARG no formato "*string**" (o asterisco não fica necessariamente após a *string*). Esse formato irá ser útil posteriormente.

Realizamos, então, um ciclo *while* para a leitura de cada linha do ficheiro TEMP.txt (já terá sido executado o comando para copiar os dados do ficheiro TABELA.txt para o ficheiro mencionado). Ao ler cada linha do ficheiro, iremos extrair a informação respetiva à primeira coluna da linha - referente à coluna COMM - para a variável $nome.

De seguida, iremos comparar a variável $nome com o nosso atual $OPTARG, tendo em conta que comparar uma *string* a, por exemplo:

- d*: retorna *true* se a *string* começar com d.
- *d: retorna *true* se a *string* terminar com d.
- *d*: retorna *true* se a *string* contiver a letra d.

Se o conteúdo da variável $nome quando comparado com o $OPTARG retornar *true*, a linha analisada é copiada integralmente para o ficheiro TABELA.txt (que no momento se encontra vazio).

```
{
while read line; do
      nome=$(awk '{printf $1}' <<< $line)
      if [[ $nome == $OPTARG ]]; then
            echo "$line" >> TABELA.txt
      fi
done
}<TEMP.txt
;;
```

Opção -s:

Quando a opção -s é passada como argumento, após a validação do seu $OPTARG, já anteriormente explicada, realizamos um ciclo *while* para a leitura de cada linha do ficheiro TEMP.txt (já terá sido executado o comando para copiar os dados do ficheiro TABELA.txt

para o ficheiro mencionado). Ao ler cada linha do ficheiro, extraímos a informação respetiva à última coluna da linha - referente à coluna DATE - para a variável $data. Tratamos a informação como fora detalhado anteriormente, portanto o código abaixo não carece de explicações.

```
cp TABELA.txt > TEMP.txt;
: > TABELA.txt
{
while read line; do
      data=$(awk '{printf $10}' <<< $line)
      data="${data//_/ }"
      mesTabela=$(awk '{printf $1}' <<< $data)
      diaTabela=$(awk '{printf $2}' <<< $data)
      horarioTabela=$(awk '{printf $3}' <<< $data)
      horarioTabela="${horarioTabela//:/ }"
      horaTabela=$(awk '{printf $1}' <<< $horarioTabela)
      minTabela=$(awk '{printf $2}' <<< $horarioTabela)
      segTabela=$(awk '{printf $3}' <<< $horarioTabela)
      if [[ "$mesTabela" == "Dez" ]] || [[ "$mesTabela" == "Dec" ]]; then
            mesTabela=12
      elif [[ "$mesTabela" == "Nov" ]]; then
            mesTabela=11
      elif [[ "$mesTabela" == "Oct" ]] || [[ "$mesTabela" == "Out" ]]; then
            mesTabela=10
      elif [[ "$mesTabela" == "Sep" ]] || [[ "$mesTabela" == "Set" ]]; then
            mesTabela=9
      elif [[ "$mesTabela" == "Ago" ]] || [[ "$mesTabela" == "Aug" ]]; then
            mesTabela=8
      elif [[ "$mesTabela" == "Jul" ]]; then
            mesTabela=7
      elif [[ "$mesTabela" == "Jun" ]]; then
            mesTabela=6
      elif [[ "$mesTabela" == "May" ]] || [[ "$mesTabela" == "Mai" ]]; then
            mesTabela=5
      elif [[ "$mesTabela" == "Apr" ]] || [[ "$mesTabela" == "Abr" ]]; then
            mesTabela=4
      elif [[ "$mesTabela" == "Mar" ]]; then
            mesTabela=3
      elif [[ "$mesTabela" == "Feb" ]] || [[ "$mesTabela" == "Fev" ]]; then
            mesTabela=2
      elif [[ "$mesTabela" == "Jan" ]]; then
            mesTabela=1
      fi
      ...
```

A opção de ordenação -s diz respeito à data mínima, o que significa que precisamos de realizar um conjunto de condições *if*, de forma a filtrar quais linhas analisadas é que possuem uma data superior à passada como argumento e, portanto, serão escritas no ficheiro TABELA.txt.

```
if [[ $mesTabela -gt $mesArg ]]; then
       echo "$line" >> TABELA.txt
else
      if [[ $mesTabela -eq $mesArg ]]; then
            if [[ $diaTabela -gt $diaArg ]]; then
                  echo "$line" >> TABELA.txt
            else
                  if [[ $diaTabela -eq $diaArg ]]; then
                        if [[ $horaTabela -gt $horaArg ]]; then
                              echo "$line" >> TABELA.txt
                        else
                              if [[ $horaTabela -eq $horaArg ]]; then
                                    if [[ $minTabela -gt $minArg ]]; then
                                          echo "$line" >> TABELA.txt
                                    else
                                          if [[ $minTabela -eq $minArg ]]; then
                                                if [[ $segTabela -ge $segArg ]]
                                                then
                                                      echo "$line" >> TABELA.txt
                                                fi
                                          fi
                                    fi
                              fi
                        fi
                  fi
            fi
      fi
fi
done
}<TEMP.txt
;;
```

Opção -e:

Quando a opção -e é passada como argumento, após a validação do seu $OPTARG, já anteriormente explicada, realizamos um ciclo *while* para a leitura de cada linha do ficheiro TEMP.txt (já terá sido executado o comando para copiar os dados do ficheiro TABELA.txt para o ficheiro mencionado). Ao ler cada linha do ficheiro, extraímos a informação respetiva à última coluna da linha - referente à coluna DATE - para a variável $data. Tratamos a informação como fora detalhado anteriormente, portanto o código abaixo não carece de explicações.

```
cp TABELA.txt > TEMP.txt;
: > TABELA.txt
{
while read line; do
      data=$(awk '{printf $10}' <<< $line)
      data="${data//_/ }"
      mesTabela=$(awk '{printf $1}' <<< $data)
      diaTabela=$(awk '{printf $2}' <<< $data)
      horarioTabela=$(awk '{printf $3}' <<< $data)
```

```
horarioTabela="${horarioTabela//:/ }"
horaTabela=$(awk '{printf $1}' <<< $horarioTabela)
minTabela=$(awk '{printf $2}' <<< $horarioTabela)
segTabela=$(awk '{printf $3}' <<< $horarioTabela)
if [[ "$mesTabela" == "Dez" ]] || [[ "$mesTabela" == "Dec" ]]; then
       mesTabela=12
elif [[ "$mesTabela" == "Nov" ]]; then
       mesTabela=11
elif [[ "$mesTabela" == "Oct" ]] || [[ "$mesTabela" == "Out" ]]; then
       mesTabela=10
elif [[ "$mesTabela" == "Sep" ]] || [[ "$mesTabela" == "Set" ]]; then
       mesTabela=9
elif [[ "$mesTabela" == "Ago" ]] || [[ "$mesTabela" == "Aug" ]]; then
       mesTabela=8
elif [[ "$mesTabela" == "Jul" ]]; then
       mesTabela=7
elif [[ "$mesTabela" == "Jun" ]]; then
       mesTabela=6
elif [[ "$mesTabela" == "May" ]] || [[ "$mesTabela" == "Mai" ]]; then
       mesTabela=5
elif [[ "$mesTabela" == "Apr" ]] || [[ "$mesTabela" == "Abr" ]]; then
       mesTabela=4
elif [[ "$mesTabela" == "Mar" ]]; then
       mesTabela=3
elif [[ "$mesTabela" == "Feb" ]] || [[ "$mesTabela" == "Fev" ]]; then
       mesTabela=2
elif [[ "$mesTabela" == "Jan" ]]; then
       mesTabela=1
fi
...
```

A opção de ordenação -e diz respeito à data máxima, o que significa que precisamos de realizar um conjunto de condições *if*, de forma a filtrar quais linhas analisadas é que possuem uma data inferior à passada como argumento e, portanto, serão escritas no ficheiro TABELA.txt.

```
if [[ $mesTabela -lt $mesArg ]]; then
       echo "$line" >> TABELA.txt
else
      if [[ $mesTabela -eq $mesArg ]]; then
            if [[ $diaTabela -lt $diaArg ]]; then
                   echo "$line" >> TABELA.txt
            else
                  if [[ $diaTabela -eq $diaArg ]]; then
                        if [[ $horaTabela -lt $horaArg ]]; then
                              echo "$line" >> TABELA.txt
                        else
                              if [[ $horaTabela -eq $horaArg ]]; then
                                    if [[ $minTabela -lt $minArg ]]; then
                                          echo "$line" >> TABELA.txt
                                    else
                                          if [[ $minTabela -eq $minArg ]]; then
```

```
                                                        if [[ $segTabela -le $segArg ]]
                                                        then
                                                              echo "$line" >> TABELA.txt
                                                        fi
                                                  fi
                                            fi
                                      fi
                                fi
                          fi
                    fi
              fi
        fi
fi
done
}<TEMP.txt
;;
```

Opção -r:

Quando a opção -r é passada como argumento, temos que analisar qual função ela irá desempenhar no código; ou seja, se ela está associada às opções -m, -t, -d ou -w e, portanto, a sua função é inverter a ordem original da opção associada (observe que, se esse for o caso, o código reverso é realizado na secção da opção associada e não na secção referente ao -r), ou se ela não está associada a nenhuma opção ou está associada às opções -c, -p, -e, -s e -u e, portanto, a sua função é ordenar a informação por ordem alfabética inversa.

Se o número total de argumentos passados for igual a 2 (dois), sabendo que um deles é o parâmetro tempo de espera, automaticamente sabemos que o primeiro é o parâmetro -r. Nesta situação, o -r não está associado a qualquer outra opção, portanto o papel que ele irá desempenhar será de organizar os dados por ordem alfabética inversa.

```
if [[ $# -eq 2 ]]; then
      sort -r TABELA.txt > TEMP.txt
      : > TABELA.txt
      cp TEMP.txt TABELA.txt
...
```

Se o número total de argumentos passados for igual a 3 (três) e estes forem passados pela ordem correta, sabemos que o -r estará associado a uma das opções -m, -t, -d e -w, pelo que não precisamos de nos preocupar com a realização de código nesta secção (o respetivo código é escrito na secção do parâmetro -m, -t, -d ou -w, dependendo a qual opção está associada).

Se o número total de argumentos passados for superior a 3 (três), teremos de descobrir se o -r está associado a uma das opções -m, -t, -d ou -w e, caso não esteja, então organizamos o ficheiro TABELA.txt por ordem alfabética inversa.

```
...
```

```
    indexR=1
    for i in "${@}"; do
          indexO=1
          for k in "${@}"; do
                if [[ "$i" == "-r" ]]; then
                      if [[ "$k" == "-m" ]] || [[ "$k" == "-t" ]] || [[ "$k" == "-d"
]] || [[ "$k" == "-w" ]]; then
                            ...
                      else
                            sort -r TABELA.txt > TEMP.txt
                            : > TABELA.txt
                            cp TEMP.txt TABELA.txt
                      fi
    ...
```

Terminada a análise de todas as opções passadas como argumento, realizamos as duas últimas linhas de código, responsáveis pela impressão da informação do ficheiro TABELA.txt com o respetivo cabeçalho.

```
    printf '%-19s %-21s %9s %15s %15s %20s %20s %20s %20s %21s\n' "COMM" "USER" "PID"
"MEM" "RSS" "READB" "WRITEB" "RATER" "RATEW" "DATE"
    cat TABELA.txt
```

7. Testes realizados

Durante o desenvolvimento do código foram realizados diversos testes de forma a validar o *script* escrito e conseguir o resultado final obtido.Nesta secção encontrará *screenshots* da informação exibida no terminal quando os argumentos passados são inválidos e válidos.

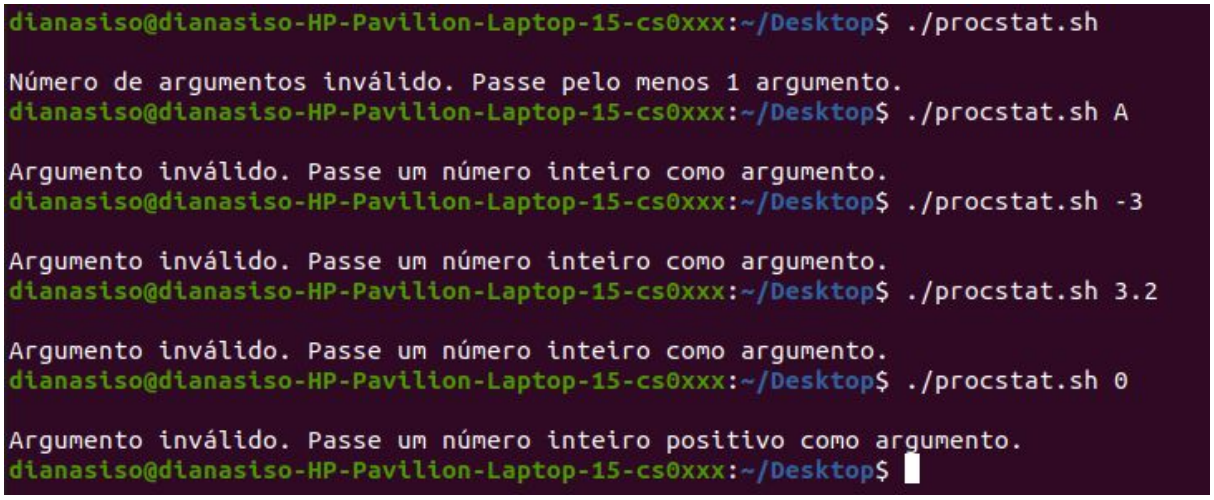

Fig. 1 – Mensagens exibidas no terminal quando o argumento referente ao tempo não é passado de forma correta.

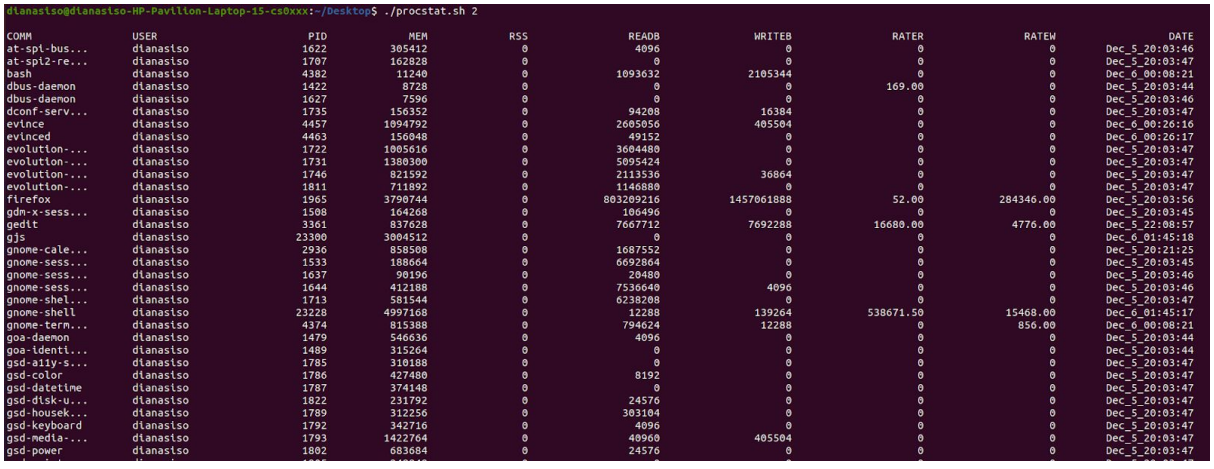

Fig. 2 – Informação exibida no terminal quando o argumento referente ao tempo é passado de forma correta.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh 2 2 2

Argumentos inválidos. Passou argumentos que não serão usados ou faltam argumentos!
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -r -r 2

Argumentos inválidos. Não introduza a mesma opção de ordenação mais do que uma vez.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -p 2

Argumentos inválidos. Passou argumentos que não serão usados ou faltam argumentos!
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$
```

Fig. 3 – Mensagens exibidas no terminal quando há argumentos extras ou argumentos em falta.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dec 32 13:40" 3

Segundo argumento inválido: passe um número compreendido entre 1 e 31.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dex 31 13:40" 3

Primeiro argumento inválido: passe um mês como primeiro argumento.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dec 31 34:40" 3

Horas inválidas: passe um número inteiro compreendido entre 0 e 24.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dec 31 4:70" 3

Minutos inválidos: passe um número inteiro compreendido entre 0 e 60.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dec 31 4:40:70" 3

Segundos inválidos: passe um número inteiro compreendido entre 0 e 60.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Dec 31 4" 3

Argumento inválido. Passe o parâmetro horário com pelo menos a informação referente às horas e minutos.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$
```

Fig. 4 – Mensagens exibidas no terminal quando as datas associadas às opções -s e -e não são válidas.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -s "Nov 20 4:21" -e "Dec 31 4:20" 3
```

COMM	USER	PID	MEM	RSS	READB	WRITEB	RATER	RATEW	DATE
at-spi-bus...	dianasiso	1622	305412	0	4096	0	0	0	Dec_5_20:03:46
at-spi2-re...	dianasiso	1707	162828	0	0	0	0	0	Dec_5_20:03:47
bash	dianasiso	4382	11240	0	1384448	5074944	0	0	Dec_6_00:08:21
dbus-daemon	dianasiso	1422	8864	0	0	0	112.66	0	Dec_5_20:03:44
dbus-daemon	dianasiso	1627	7596	0	0	0	0	0	Dec_5_20:03:46
dconf-serv...	dianasiso	1735	156352	0	94208	16384	0	0	Dec_5_20:03:47
evince	dianasiso	4457	1095008	0	2605056	405504	0	0	Dec_6_00:26:16
evinced	dianasiso	4463	156048	0	49152	0	0	0	Dec_6_00:26:17
evolution-...	dianasiso	1722	1005616	0	3604480	0	0	0	Dec_5_20:03:47
evolution-...	dianasiso	1731	1380300	0	5095424	0	0	0	Dec_5_20:03:47
evolution-...	dianasiso	1746	821592	0	2113536	36864	0	0	Dec_5_20:03:47
evolution-...	dianasiso	1811	711892	0	1146880	0	0	0	Dec_5_20:03:47
firefox	dianasiso	1965	3823920	0	803209216	1504559104	26334.33	9642.33	Dec_5_20:03:56
gdm-x-sess...	dianasiso	1508	164268	0	106496	0	0	0	Dec_5_20:03:45
gedit	dianasiso	3361	838052	0	7667712	8556544	10661.33	3000.00	Dec_5_22:08:57
gjs	dianasiso	96170	2930780	0	0	0	0	0	Dec_6_04:18:19
gnome-cale...	dianasiso	2936	858508	0	1687552	0	0	0	Dec_5_20:21:25
gnome-sess...	dianasiso	1533	188664	0	6692864	0	0	0	Dec_5_20:03:45
gnome-sess...	dianasiso	1637	90196	0	20480	0	0	0	Dec_5_20:03:46
gnome-sess...	dianasiso	1644	412188	0	7536640	4096	0	0	Dec_5_20:03:46
gnome-shel...	dianasiso	1713	581676	0	6238208	0	0	0	Dec_5_20:03:47
gnome-shell	dianasiso	95870	4730612	0	0	94208	446796.00	12733.33	Dec_6_04:18:18
gnome-term...	dianasiso	4374	815388	0	794624	12288	0	637.33	Dec_6_00:08:21
goa-daemon	dianasiso	1479	546636	0	4096	0	0	0	Dec_5_20:03:44
goa-identi...	dianasiso	1489	315264	0	0	0	0	0	Dec_5_20:03:44
gsd-a11y-s...	dianasiso	1785	310188	0	0	0	0	0	Dec_5_20:03:47
gsd-color	dianasiso	1786	427480	0	8192	0	0	0	Dec_5_20:03:47
gsd-datetime	dianasiso	1787	374148	0	0	0	0	0	Dec_5_20:03:47
gsd-disk-u...	dianasiso	1822	231792	0	24576	0	0	0	Dec_5_20:03:47
gsd-housek...	dianasiso	1789	312256	0	303104	0	4048.33	0	Dec_5_20:03:47

Fig. 5 – Informação exibida no terminal quando as datas associadas às opções -s e -e são válidas.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -r -m 1

Argumentos inválidos! Verifique se passou os argumentos -m, -t, -d ou -w antes do -r ou se passou argumentos válidos.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -r -t 1

Argumentos inválidos! Verifique se passou os argumentos -m, -t, -d ou -w antes do -r ou se passou argumentos válidos.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -r -d 1

Argumentos inválidos! Verifique se passou os argumentos -m, -t, -d ou -w antes do -r ou se passou argumentos válidos.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -w -d 1

Argumentos inválidos! Não pode passar a opção -w com as opções -m, -t ou -d.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -m -w 1

Argumentos inválidos! Não pode passar a opção -m com as opções -t, -d ou -w.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -t -w 1

Argumentos inválidos! Não pode passar a opção -t com as opções -m, -d ou -w.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -d -m 1

Argumentos inválidos! Não pode passar a opção -d com as opções -m, -t ou -w.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -r -e "Dec 31 12:30" -d 1

Argumentos inválidos! Verifique se passou os argumentos -m, -t, -d ou -w antes do -r.
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -m -e "Dec 31 12:30" -r 1

Argumentos inválidos! Verifique se as opções -m/-t/-d/-w estão exatamente antes do -r
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$
```

Fig. 6 - Mensagens exibidas no terminal quando as opções -m, -t, -d, -w e -r não são válidas.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -m -r -e "Dec 31 12:30" 1

COMM            USER              PID          MEM        RSS          READB          WRITEB          RATER          RATEW             DATE
dbus-daemon     dianasiso        1627         7596          0              0               0              0              0  Dec_5_20:03:46
dbus-daemon     dianasiso        1422         8864          0              0               0              0              0  Dec_5_20:03:44
p               dianasiso      264225         9624          0          24576           28672       63722400.00      969473.00  Dec_6_04:33:53
b               dianasiso        4382        11240          0        1601536        15847424              0              0  Dec_6_00:08:21
gnome-sess...   dianasiso        1637        90196          0          20480               0              0              0  Dec_5_20:03:46
evinced         dianasiso        4463       156048          0          49152               0              0              0  Dec_6_00:26:17
dconf-serv...   dianasiso        1735       156352          0          94208           16384              0              0  Dec_5_20:03:47
at-spi2-re...   dianasiso        1707       162828          0              0               0              0              0  Dec_5_20:03:47
gvfsd-meta...   dianasiso        2282       163124          0          90112         2334720        6055.00         480.00  Dec_5_20:04:47
ibus-engin...   dianasiso      169852       163188          0              0               0              0              0  Dec_6_04:24:24
ibus-memconf    dianasiso      169548       163200          0              0               0              0              0  Dec_6_04:24:23
gdm-x-sess...   dianasiso        1508       164268          0         106496               0              0              0  Dec_5_20:03:45
rddprocess      dianasiso        2669       187472          0          36864               0              0              0  Dec_5_20:07:30
gnome-sess...   dianasiso        1533       188664          0        6692864               0              0              0  Dec_5_20:03:45
ibus-x11        dianasiso      169553       194100          0              0               0              0              0  Dec_6_04:24:23
gsd-disk-u...   dianasiso        1822       231792          0          24576               0              0              0  Dec_5_20:03:47
gsd-screen...   dianasiso        1808       235780          0              0               0              0              0  Dec_5_20:03:47
xdg-permis...   dianasiso        1711       235936          0           8192               0              0              0  Dec_5_20:03:47
gvfs-goa-v...   dianasiso        1474       236144          0              0               0              0              0  Dec_5_20:03:44
gvfs-mtp-v...   dianasiso        1470       236812          0              0               0              0              0  Dec_5_20:03:44
ibus-portal     dianasiso      169555       236976          0              0               0              0              0  Dec_6_04:24:23
gvfs-gphot...   dianasiso        1466       238344          0              0               0              0              0  Dec_5_20:03:44
gvfsd           dianasiso        1442       240080          0         102400               0              0              0  Dec_5_20:03:44
gsd-print-...   dianasiso        1805       248848          0              0               0              0              0  Dec_5_20:03:47
ibus-exten...   dianasiso      169549       271268          0              0               0              0              0  Dec_6_04:24:23
at-spi-bus...   dianasiso        1622       305412          0           4096               0              0              0  Dec_5_20:03:46
gsd-a11y-s...   dianasiso        1785       310188          0              0               0              0              0  Dec_5_20:03:47
ibus-daemon     dianasiso      169544       311676          0              0            4096              0              0  Dec_6_04:24:23
gsd-housek...   dianasiso        1789       312256          0         303104               0              0              0  Dec_5_20:03:47
gvfsd-trash     dianasiso        1773       314480          0         442368               0              0              0  Dec_5_20:03:47
gsd-wwan        dianasiso        1823       314512          0          40960               0              0              0  Dec_5_20:03:47
gvfs-udisk...   dianasiso        1454       314684          0              0               0              0              0  Dec_5_20:03:44
goa-identi...   dianasiso        1489       315264          0              0               0              0              0  Dec_5_20:03:44
gsd-smartc...   dianasiso        1816       315688          0              0               0              0              0  Dec_5_20:03:47
gvfs-afc-v...   dianasiso        1460       316996          0              0               0              0              0  Dec_5_20:03:44
gsd-sound       dianasiso        1819       319880          0              0               0              0              0  Dec_5_20:03:47
gsd-wacom       dianasiso        1821       342000          0         245760               0              0              0  Dec_5_20:03:47
```

Fig. 7 - Exemplo de informação exibida no terminal quando as opções -m, -t, -d, -w e -r são válidas.

```
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -u dianasiso -c "d.*"  3

COMM            USER              PID          MEM        RSS          READB          WRITEB          RATER          RATEW             DATE
dbus-daemon     dianasiso        1422         8864          0              0               0         112.66              0  Dec_5_20:03:44
dbus-daemon     dianasiso        1627         7596          0              0               0              0              0  Dec_5_20:03:46
dconf-serv...   dianasiso        1735       156352          0          94208           16384              0              0  Dec_5_20:03:47
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$ ./procstat.sh -u dianasiso -c "*web*"  3

COMM            USER              PID          MEM        RSS          READB          WRITEB          RATER          RATEW             DATE
webcontent      dianasiso        2043      3877204          0       36794368               0          61.66          61.66  Dec_5_20:03:57
webcontent      dianasiso        2211      2763040          0       12066816          196608           9.66           9.66  Dec_5_20:04:08
webcontent      dianasiso        2382      2931716          0       21872640               0          41.66          41.66  Dec_5_20:06:27
webcontent      dianasiso       41329      3038900          0          57344               0          95.66          95.66  Dec_6_01:47:40
webcontent      dianasiso       41715      2397640          0              0               0              0              0  Dec_6_02:38:21
webextenst...   dianasiso        2092      2426500          0         385024               0           6.66           6.66  Dec_5_20:03:58
dianasiso@dianasiso-HP-Pavilion-Laptop-15-cs0xxx:~/Desktop$
```

Fig. 8 - Exemplo de informação exibida no terminal quando introduzidas as opções -u e -c.

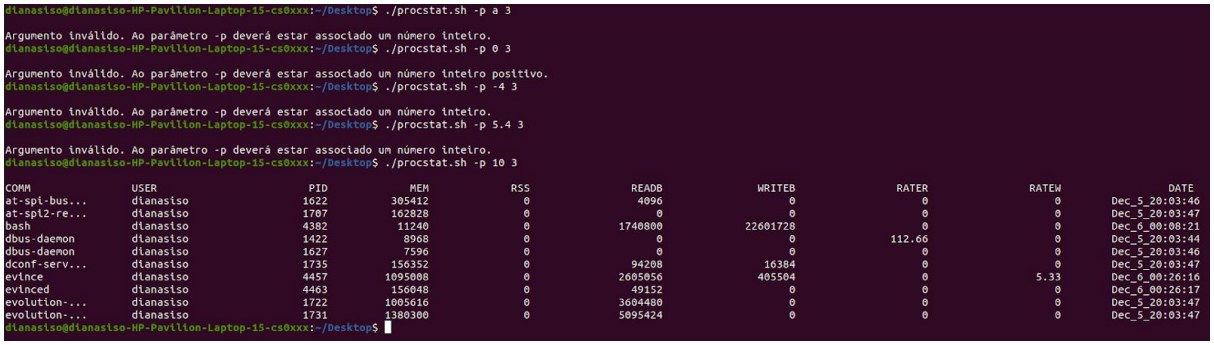

Fig. 9 – Mensagens exibidas no terminal quando a opção -p não é válida e informação emitida no terminal quando o parâmetro associado à opção -p é válida.

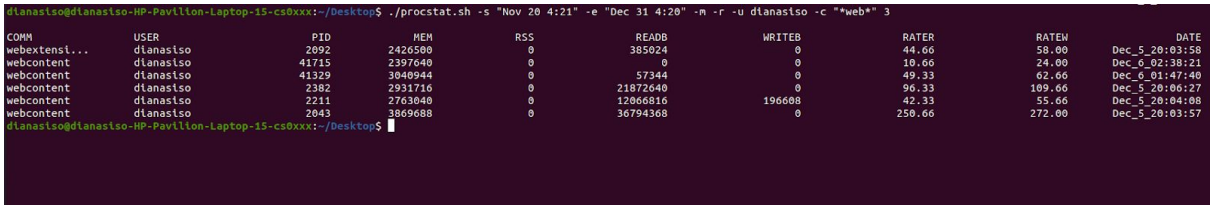

Fig. 10 – Informação exibida no terminal quando são passadas várias opções de filtragem que podem coexistir juntas.