universidade de aveiro

Sistemas Operativos

Estatísticas de processos em bash

Trabalho n.º 1

2020/2021

Diogo Pereira Henriques da Cruz, nº 98595, P2
Mariana Cabral Silva Silveira Rosa, nº 98390, P5

Conteúdos

Introdução

O conceito deste 1º trabalho prático incide sobre as estatísticas da memória dos diferentes processos, da quantidade de I/O que uma seleção de processos está a efetuar durante a execução nos nossos computadores. Como aprendemos na unidade curricular Sistemas Operativos, um processo é um programa em execução, que pode ser criado aquando da inicialização do sistema, de uma chamada ao sistema por um processo ou pelo pedido de um utilizador.

Assim, o objetivo principal deste trabalho é desenvolver um *script* na linguagem Bash que nos permita visualizar as diferentes características dos variados processos. Com as informações fornecidas nas aulas teóricas e práticas desta unidade curricular e alguma pesquisa foi possível desenvolver o *script* **procstat.sh.**

Abordagem ao problema

Em primeiro lugar, para elaborar este *script* precisamos de compreender o objetivo indicado pelo docente, assim, reunir um conjunto de informações sobre como podemos alcançar com sucesso o *script* : pretendemos visualizar a quantidade de memória total e da memória física, o número total de bytes de I/O e a taxa de leitura/escrita (em bytes por segundo) dos processos selecionados nos últimos **s** segundos. Logicamente, compreendemos que este último parâmetro mencionado (o número de segundos) terá de ser obrigatório quando o utilizador recorre ao nosso *script.*

Para tornar o *script* mais objetivo e simples de utilizar, disponibilizamos algumas opções que mais tarde iremos explicar em detalhe.

Estruturação do código

Em 1º lugar, começamos por definir algumas variáveis (Fig. 1) que irão ser utilizadas ao longo do código, nomeadamente para o uso das diferentes opções, as "flags". De seguida, construímos duas strings utilizadas para a ordenação das colunas, estas strings serão alteradas consoante o critério do utilizador. Definimos também que a ordem padrão seria por ordem crescente.

Declaramos o *array procs* que mais tarde irá guardar as informações dos nossos processos.

```
#!/bin/bash

flagU=0;
flagC=0;
flagP=0;
flagS=0;
flagE=0;

flagSort=0;          # verificaçao
flagReverse=0;       # verificar default

sort="sort "
order=" -n "         # ordem por default


declare -a procs; #declarar array
```

Fig. 1 - Definição inicial

Posteriormente declaramos uma mensagem para o *output* do utilizador com as instruções para o uso correto do programa e das suas diferentes opções.

```
message(){

    echo "
        ------------------------------------------- OPÇÕES VÁLIDAS -------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

    Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

    Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
            -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r      : reverse order
                -m      : sorts on MEM
                -t      : sorts on RSS
                -d      : sorts on RATER
                -w      : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'

        "
}
```

Fig. 2 - Instruções do *script*.

Em relação ao uso das diferentes opções e das suas validações iremos explicar mais à frente, uma vez que é necessário perceber o funcionamento do nosso código e de como obtemos todas as informações para a construção da nossa tabela de estatísticas (tal como nos foi pedido no enunciado deste trabalho prático; Fig. 3).

```
$./procstat.sh 10
COMM      USER       PID     MEM     RSS   READB   WRITEB      RATER      RATEW          DATE
bash      nlau     10174 3724656  452880   24690    58597 3770687.30   14239.00 Sep 12 11:45
dice      sop0200   5036  123770   45005 2914000     1356    4022.70    5114.70 Sep 23 18:14
dropbox   sop0100   2636 3288072  335932  918784  1356760   40022.70   31114.50 Sep 19 08:49
```

Fig. 3 - Tabela de estatísticas que teremos de reproduzir

Definição de cada coluna:

- COMM - Nome do processo;
- USER - Quem criou o processo;
- PID - Identificador único do processo ativo;
- MEM - Quantidade de memória virtual ocupada pelo processo;
- RSS - Quantidade de memória física RAM ocupada pelo processo;
- READB - Quantidade de bytes I/O escritos pelo processo;
- WRITEB - Quantidade de bytes I/O lidos pelo processo;
- RATER - Taxa de bytes lidos por segundo;
- RATEW - Taxa de bytes escritos por segundo;
- DATE - Data da criação do processo.

Explicando de forma simples o que foi feito foi colocar-mo-nos na diretoria **/proc** usando o comando **cd /proc** e iniciámos dois ciclos *for* para percorrer todos os processos denominados com números (pois existem outros que não nos interessam), uma vez que iremos precisar de calcular uma taxa onde é usado o comando **sleep $segundos.** Simplificadamente, após ler os primeiros valores *READB* e *WRITEB* no 1º *for*, vamos fazer um **sleep** e de seguida entramos no 2º *for* onde iremos calcular os novos valores de *READB* e *WRITEB* e calculamos a taxa através desses 2 valores e do número de segundos usado no comando **sleep**.

Para ler os valores dentro destes ciclos for precisamos de verificar que existe a diretoria e se existe certificar que tem as subdiretorias **/comm**, **/status**, e **/io** da qual esta última também temos de ter permissão de leitura. Desta forma temos acesso a todos os dados que precisamos para aceder à informação chave para construir a tabela.

Para o 1º loop criamos 3 *arrays*:

- array *pid*: guarda o valor de todos os PIDs a correr no momento
- array *readb*: guarda a quantidade de bytes I/O escrito por cada processo
- array *writeb*: guarda a quantidade de bytes I/O lido por cada processo

Estes arrays são criados pois são dados importantes que queremos guardar para mais tarde usar no 2º *for*, após executarmos o comando *sleep*.
Chegando ao 1º *for* (Fig. 4) usamos comandos como **cat**, **grep** (**-i "<string> "** para tirar a variável desejada e **-o -E '[0-9]+'** para retirar só o seu valor numeral) e **tr (**-dc '0-9' para retirar o valor do processo único) para retirar os valores desejáveis para cada variável e guardamos em cada array correspondente.

```
235  declare -a readb; #declarar array
236  declare -a writeb; #declarar array
237  declare -a pid; #declarar array
238
239  cd /proc
240  for file in *[0-9] ; do
241      if [ -f "$file/comm" ]; then
242          if [ -f "$file/io" ]; then
243              if [ -f "$file/status" ]; then
244                  if [ -r "$file/io" ]; then
245
246                      PID=$(cat $file/status | grep -w Pid | tr -dc '0-9')
247                      pid+=($PID)
248                      READB=$(cat $file/io | grep -i "rchar" | grep -o -E '[0-9]+')
249                      readb+=($READB)
250                      WRITEB=$(cat $file/io | grep -i "wchar" | grep -o -E '[0-9]+')
251                      writeb+=($WRITEB)
252
253                  fi
254              fi
255          fi
256      fi
257  done
258
```

Fig. 4 - 1º ciclo *for*

De seguida usamos o comando **sleep $segundos** onde **segundos** é o valor posto por quem está a usar este script . O comando **sleep** que faz com que não haja criação de novos processos naquele intervalo de tempo, voltamos a relembrar, o número de segundos é um parâmetro obrigatório. Definimos também 3 "contadores": **i** que é o número de processos total, **cc** que vai ser o número de processos tirados quando o utilizador usa a opção -c e **se** que também vai guardar o número de processos quando utilizador limita os processos por datas (-s e/ou -e) Estes contadores vão ser usados mais tarde a lidar com exceções. Chegando ao 2º *for* (*Fig.*5) criamos uma variável para cada coluna da tabela e para cada valor que queremos saber e associamos um valor facilmente usando comandos como **cat,**

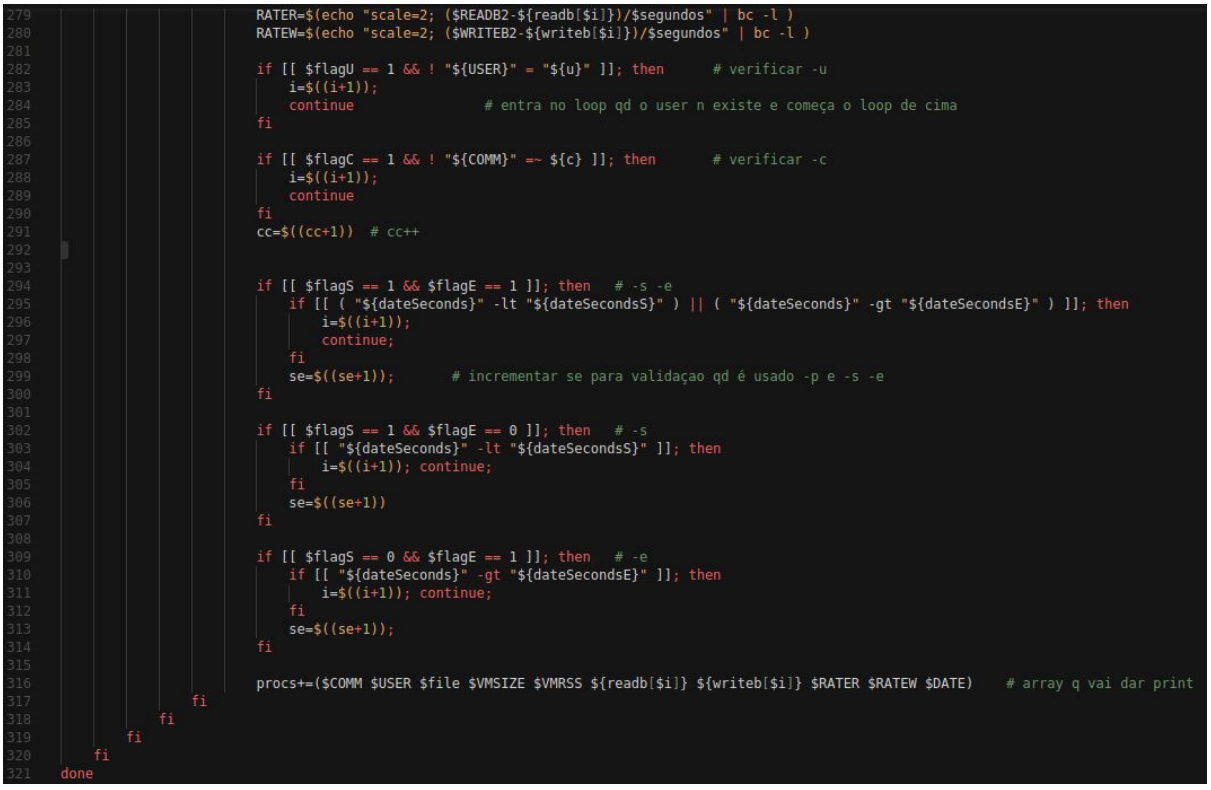
grep, **ps**, **awk,** entre outros que permitem obter a informação que desejamos saber associada a cada caso.

Executamos um ciclo *for* que garante que vão ser só lidos os processos mostrados no 1º ciclo *for*, se um processo entretanto acabar entre o 1º *for* e o sleep este não será contado para o novo *for loop*. Voltamos a fazer as mesmas verificações de **/comm**, **/status**, e **/io** mas desta vez só levando em conta os processos lidos anteriormente, fazendo desta forma o código mais eficaz pois não tem que voltar a ler todos os processos atuais novamente e procurar quais são aqueles que existiam anteriormente ou que deixaram de existir. Guardamos os valores necessários nas variáveis **COMM**, **VMSIZE, VMRSS, USER, DATE, RATER, RATEW** que serão informações importantes para a tabela de valores, entre outras variáveis auxiliares. Se tudo se verificar vamos calcular a taxa de bits lidos e escritos por segundo ao subtrair os dados depois do sleep (**READB2** e **WRITEB2**), os dados antes do sleep (**readb[i]** e **writeb[i]**)e dividir pelo número de segundos passados(**$segundos**), iremos guardar estes valores em **RATER** e **RATEW**. Após recolhermos todas as informações necessárias, adicionamos aos *array* **procs**, processo a processo, as informações que iremos imprimir na tabela. Ao acabar o loop volta ao for e ao ver que a variável **i** é menor que o número de processos lidos no 1º *for* então incrementamos a variável **i**, e recomeçamos os ciclos em busca de informações de um novo processo.

```
259  sleep $segundos
260  i=0;    # incrementar dps os arrays
261  se=0;   # ver a quantidade de processoes dentro dos parametros das datas
262  cc=0;   # contador para o numero de comms q contem a letras especificada pelo utilizador (usar isto com as datas)
263
264  cd /proc
265  for (( i=0; i<${#pid[@]}; i++ )); do        # para todos os processos lidos no 1º loop
266      file="${pid[i]}"
267      if [ -f "$file/comm" ]; then
268          if [ -f "$file/io" ]; then
269              if [ -f "$file/status" ]; then
270                  if [ -r "$file/io" ]; then      # verificaçoes todas para se poder ler o file
271
272                          COMM=$(cat $file/comm | tr " " "_")
273                          VMSIZE=$(cat $file/status | grep -i "VmSize" | grep -o -E '[0-9]+')
274                          VMRSS=$(cat $file/status | grep -i "VmRSS" | grep -o -E '[0-9]+')
275                          USER=$(ps -o uname= -p "${pid[$i]}" )
276                          DATE=$(LC_ALL=EN_us.utf-8 ls -ld "${file}" | awk '{print $6 " " $7 " " $8 }')
277                          dateSeconds=$(date -d "$DATE:00 2020" "+%s");
278                          READB2=$(cat $file/io | grep -i "rchar" | grep -o -E '[0-9]+')
279                          WRITEB2=$(cat $file/io | grep -i "wchar" | grep -o -E '[0-9]+')
280                          RATER=$(echo "scale=2; ($READB2-${readb[$i]})/$segundos" | bc -l )
281                          RATEW=$(echo "scale=2; ($WRITEB2-${writeb[$i]})/$segundos" | bc -l )
```

Fig. 5 - 2º ciclo *for*

Tratamento das opções



```
                    RATER=$(echo "scale=2; ($READB2-${readb[$i]})/$segundos" | bc -l )
                    RATEW=$(echo "scale=2; ($WRITEB2-${writeb[$i]})/$segundos" | bc -l )

                    if [[ $flagU == 1 && ! "${USER}" = "${u}" ]]; then      # verificar -u
                        i=$((i+1));
                        continue                # entra no loop qd o user n existe e começa o loop de cima
                    fi

                    if [[ $flagC == 1 && ! "${COMM}" =~ ${c} ]]; then       # verificar -c
                        i=$((i+1));
                        continue
                    fi
                    cc=$((cc+1))  # cc++


                    if [[ $flagS == 1 && $flagE == 1 ]]; then   # -s -e
                        if [[ ( "${dateSeconds}" -lt "${dateSecondsS}" ) || ( "${dateSeconds}" -gt "${dateSecondsE}" ) ]]; then
                            i=$((i+1));
                            continue;
                        fi
                        se=$((se+1));       # incrementar se para validaçao qd é usado -p e -s -e
                    fi

                    if [[ $flagS == 1 && $flagE == 0 ]]; then   # -s
                        if [[ "${dateSeconds}" -lt "${dateSecondsS}" ]]; then
                            i=$((i+1)); continue;
                        fi
                        se=$((se+1))
                    fi

                    if [[ $flagS == 0 && $flagE == 1 ]]; then   # -e
                        if [[ "${dateSeconds}" -gt "${dateSecondsE}" ]]; then
                            i=$((i+1)); continue;
                        fi
                        se=$((se+1));
                    fi

                    procs+=($COMM $USER $file $VMSIZE $VMRSS ${readb[$i]} ${writeb[$i]} $RATER $RATEW $DATE)    # array q vai dar print
                fi
            fi
        fi
    fi
done
```

Fig. 6 - Tratamentos das opções através das *flags*

O nosso programa suporta diferentes opções para retornar estatísticas mais precisas para o objetivo do utilizador. Para tratarmos as opções inseridas pelo utilizador, utilizamos a estrutura *getopts* que facilita bastante a construção do código. Consiste, basicamente, em alterar o valor 0 para 1 de uma "*flag*" para sinalizar que aquela opção foi ativada (Fig. 6). Assim, através de um conjunto de ciclos de condições conseguimos alcançar o nosso objetivo (Fig. 7).

```
46
47  while getopts ":c:u:s:e:p:rmdtw" options; do
48      case "${options}" in
49      u) # selecção dos processos ser realizada através do nome do utilizador
50          u="${OPTARG}"
51
52          if [[ $flagU == 1 ]]; then
53              echo "ERRO: -u <OPTION> só pode ser selecionado uma vez!"
54              exit 1
55          fi
56
57          flagU=1;
58      ;;
59
```

Fig. 7 - Tratamento das opções; *if*'s

Existem opções que contém argumentos obrigatórios (OPTARG) tal como a escolha da quantidade de processos a visualizar ou mostrar os processos de um dado utilizador, outras não, como as opções de ordenação. Para concretizar esta distinção escrevemos a letra da opção seguida de ":" , significando que aquela opção irá aceitar argumentos.

O nosso programa suporta diferentes opções para retornar estatísticas mais precisas para o objetivo do utilizador. Mas como funcionam?

Filtros de procura:

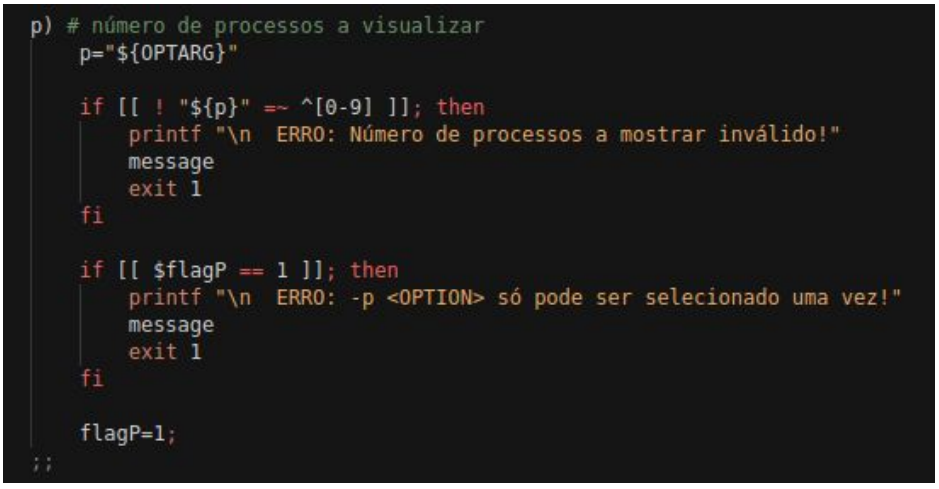
- **-u** : Requer um argumento obrigatório, apenas pode ser utilizado uma vez e filtra os processos executados apenas pelo nome de um utilizador escolhido pelo utilizador do programa (imagem seguinte refere à opção *-u* do *getopts*).

```
u) # selecção dos processos ser realizada através do nome do utilizador
    u="${OPTARG}"

    if [[ $flagU == 1 ]]; then
        echo "ERRO: -u <OPTION> só pode ser selecionado uma vez!"
        exit 1
    fi

    flagU=1;
;;
```

Através de uma condição *if*, se a *flag* estiver ativa, ele vai verificar se o processo que está a analisar corresponde à autoria daquele utilizador, se não corresponder incrementa a variável **i**, correspondente ao próximo processo e segue em frente não chegando a adicionar aquele processo no *array* **procs** (imagem seguinte mostra a validação da variável **USER,** se esta corresponde ao valor usado pelo utilizador ao correr o script então o **continue** não é executado e o programa salta o *if*, adicionando o valor de **USER** ao array *procs*).

```
if [[ $flagU == 1 && ! "${USER}" = "${u}" ]]; then      # verificar -u
    i=$((i+1));
    continue                # entra no loop qd o user n existe e começa o loop de cima
fi
```

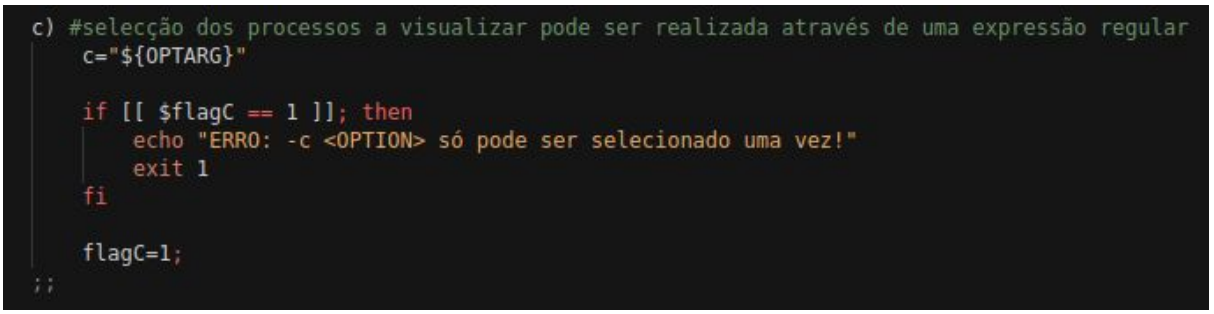
- **-p**: O utilizador escolhe quantos processos quer visualizar e através do comando **head** conseguimos visualizar "p" processos. Verifica se **$OPTARG** é um número válido e também só pode ser usado uma vez por cada vez que se corre o *script* (imagem seguinte mostra a opção *-p* em *getopts*)



```
p) # número de processos a visualizar
    p="${OPTARG}"

    if [[ ! "${p}" =~ ^[0-9] ]]; then
        printf "\n  ERRO: Número de processos a mostrar inválido!"
        message
        exit 1
    fi

    if [[ $flagP == 1 ]]; then
        printf "\n  ERRO: -p <OPTION> só pode ser selecionado uma vez!"
        message
        exit 1
    fi

    flagP=1;
;;
```

- **-c**: Seleciona os processos iniciados por uma certa expressão/letra/número através de uma expressão regular, filtrando a coluna do **COMM**.

> Note que todas estas opções requerem um argumento obrigatório e o código assim o exige, pois fizemos validações para incentivar o utilizador a escrever exatamente o que era necessário (imagem seguinte mostra a validação de *-c* em *getopts*).



```
c) #selecção dos processos a visualizar pode ser realizada através de uma expressão regular
    c="${OPTARG}"

    if [[ $flagC == 1 ]]; then
        echo "ERRO: -c <OPTION> só pode ser selecionado uma vez!"
        exit 1
    fi

    flagC=1;
;;
```

Semelhante ao *if* para validar o utilizador aqui usamos o mesmo método, se o input do utilizador após o *-c* existir na coluna de COMM então ignora o *if* e adiciona ao array *procs*. Se ignora então incrementa um valor no contador **cc** que irá contar quantos processos finais são guardados. (imagem seguinte é a validação de $OPTARG no 2º *for*).

```
if [[ $flagC == 1 && ! "${COMM}" =~ ${c} ]]; then       # verificar -c
    i=$((i+1));
    continue
fi
cc=$((cc+1))  # cc++
```

Especificar o período temporal:

- **-s**: Seleciona os processos iniciados depois da data escolhida pelo utilizador e faz *output* de todos os processos a partir daí.

- **-e**: Seleciona todos os processos que tem a sua data de início anterior da data escolhida pelo utilizador e não faz *output* dos processos posteriores.

```
107     s) #data minima
108         dreg="[A-Za-z]{3} ([0-2][1-9]|[3][0-1]) ([0-1][0-9]|[2][0-4]):[0-5][0-9]"
109         if [[ $OPTARG =~ $dreg ]]; then
110             flagS=1;
111             dateSecondsS=$(date -d "$OPTARG" "+%s");
112             #echo $dateSecondsS
113         else
114             printf "\n  ERRO: Por favor escreva a data no formato 'MÊS DIA HORA' (ex: Jan 01 00:00)!"
115             message
116             exit 1
117         fi
118     ;;
119
120     e) # data máxima
121         dreg="[A-Za-z]{3} ([0-2][1-9]|[3][0-1]) ([0-1][0-9]|[2][0-4]):[0-5][0-9]"
122         if [[ $OPTARG =~ $dreg ]]; then
123             flagE=1;
124             dateSecondsE=$(date -d "$OPTARG" "+%s");
125             #echo $dateSecondsE
126         else
127             printf "\n  ERRO: Por favor escreva a data no formato 'MÊS DIA HORA' (ex: Jan 01 00:00)!"
128             message
129             exit 1
130         fi
131     ;;
```

Notando que os argumentos têm de ser num formato específico (“Mês dia hora:minuto”, sendo que temos validações para verificar se acontece esse erro. O utilizador pode usar qualquer um destes comandos, em separado ou combinado.

Temos 3 *ifs* dentro do 2º *for* para a verificação das datas, uma para quando ambas datas forem especificadas, e os outros dois apenas quando uma delas é identificada. A maneira como funcionam de geral forma é igual a verificações anteriores, se a data do processo atual for maior que a data máxima ou inferior ao da data mínima este irá fazer com que entre no *if* e que faça **continue**, não adicionando o processo com aquela data ao array *procs*, ignorando assim o resto do código abaixo e começando o *if* de novo mas não antes de incrementar o **i.** Quando a data é válida então não entra no *if* e incrementa o valor de **se** que funciona como o contador de quantos processos vão ser válidos com as opções selecionadas das datas.

```
if [[ $flagS == 1 && $flagE == 1 ]]; then   # -s -e
    if [[ ( "${dateSeconds}" -lt "${dateSecondsS}" ) || ( "${dateSeconds}" -gt "${dateSecondsE}" ) ]]; then
        i=$((i+1));
        continue;
    fi
    se=$((se+1));       # incrementar se para validaçao qd é usado -p e -s -e
fi

if [[ $flagS == 1 && $flagE == 0 ]]; then   # -s
    if [[ "${dateSeconds}" -lt "${dateSecondsS}" ]]; then
        i=$((i+1)); continue;
    fi
    se=$((se+1))
fi

if [[ $flagS == 0 && $flagE == 1 ]]; then   # -e
    if [[ "${dateSeconds}" -gt "${dateSecondsE}" ]]; then
        i=$((i+1)); continue;
    fi
    se=$((se+1));
fi
```

Ordenação:

- **-r**: Ao utilizar esta opção alteramos a ordem *default* crescente para ordem decrescente. Para isso, substituímos na *string* **ordem** "n" por "rn", para ordenar do número mais pequeno ao maior (ordem crescente).

```
r) #reverse passa de decrescente para crescente

    if [[ $flagReverse == 0 ]]; then
        order=" -n "            # substituir -rn por -n
        flagReverse=1
    else
        printf "\n  ERRO: Só pode utilizar '-r' no máximo uma vez!"
        message
        exit 1
    fi
;;
```

- **-m**: Ativa a flag de ordenação (flagSort) se esta ainda não tiver sido ligada e ordena a memória virtual (**MEM**) por ordem decrescente.

- **-t** : Ativa a flag de ordenação (flagSort) se esta ainda não tiver sido ligada e ordena a memória física (**RSS**) por ordem decrescente.

- **-d**: Ativa a flag de ordenação (flagSort) se esta ainda não tiver sido ligada e ordena por ordem decrescente o **RATER**, que iremos explicar posteriormente.

- **-w**: Ativa a flag de ordenação (flagSort) se esta ainda não tiver sido ligada e ordena por ordem decrescente o **RATEW**, que iremos explicar posteriormente.

Note que não podemos utilizar duas ordenações em simultâneo, uma vez que a *flagSort* só é sinalizada quando se encontra inativa. Apenas podemos combinar uma ordenação de uma coluna (opções -m, -t, -d e -w) com o -r.

```
133     m) #sort na memória
134         if [[ $flagSort == 0 ]]; then
135             flagSort=1;
136             sort+=" -k4 ";
137             order=" -n "
138         else
139             printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
140             message
141             exit 1
142         fi
143         ;;
144     t)  # sort on RSS
145 
146         if [[ $flagSort == 0 ]]; then
147             flagSort=1;
148             sort+=" -k5 ";
149             order=" -n "
150         else
151             printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
152             message
153             exit 1
154         fi
155         ;;
156     d)  # sort on RATER
157 
158         if [[ $flagSort == 0 ]]; then
159             flagSort=1;
160             sort+=" -k8 ";
161             order=" -n "
162         else
163             printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
164             message
165             exit 1
166         fi
167 
168         ;;
169 
170     w)  # sort on RATEW
171 
172         if [[ $flagSort == 0 ]]; then
173             flagSort=1;
174             sort+=" -k9 ";
175             order=" -n "
176         else
177             printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
178             message
179             exit 1
180         fi
181 
```

Validação de opções de número de segundos e outras:

Última opção de *getopts* é ativada quando não são respeitadas as regras do *getopts*. Associado às funcionalidades de *getopts* é nos possível introduzir a linha de código **`shift $((OPTIND-1))`** que permite avaliar a próxima opção, caso haja alguma.

```
184       *) #colocando uma opção inválida ou não colocando um argumento onde era obrigatório, dá print da mensagem '' e termina
185           printf "\n  ERRO: Por favor introduza uma expressão válida!"
186           message
187           exit 1
188           ;;
189
190       esac
191   done
192   shift $((OPTIND-1))
193
194   # algumas validações
```

Em primeiro lugar para o código ser executado, como já foi referido anteriormente, é obrigatório introduzir um número de segundos, por isso duas verificações são necessárias a fazer: se este foi de facto introduzido e verificar se o que foi introduzido é um número através de uma expressão regular. Também verificamos se é o único número passado, pois qualquer outro número que apareça após a chamada do script só pode fazer parte de uma das opções de *getopts*.

```
216   re='^[0-9]+([.][0-9]+)?$'
217   if [[ ${@: -1} =~ $re ]]; then      # verificar se o ultimo argumento é um numero
218       segundos=${@: -1}
219   else
220       printf "\n  ERRO: Por favor introduza uma expressão válida!"
221       message
222       exit 1
223   fi
224
225   if [[ ${#@} -gt 1 ]]; then      # verificar q o unico argumento excluindo as opçoes é o $segundos
226       printf "\n  ERRO: Por favor introduza uma expressão válida!"
227       message
228       exit 1
229   fi
```

À medida que o utilizador explora o nosso *script* é muito provável que cometa erros ao navegar pelas diversas opções, assim sempre que alguma opção não é válida, o programa terminará com uma mensagem de erro associada à falha cometido e imprime as instruções (*message*).

Aqui encontramos mais 4 validações entre o *getopts* e o 1º *for*.

O 1º *if* verifica se há só **$segundos** e um tipo de *sort* (-m, -t, -d e -w), se sim então dá print da tabela por ordem alfabética.

O 2º serve para verificar se é selecionado apenas **$segundos** e *-r*, se se verificar então vai mudar a ordem da coluna de -n (valor por default) para -rn para a coluna **COMM** estar de Z-A.

O 3º *if* verifica se os únicos valores postos foi a opção *-p* e *-r*, caso se verifique a ordem será -rn para ficar por ordem contrária do alfabeto.

O último *if* garante que quando o utilizador usa ambas opções *-s* e *-e* o valor da data mínima não pode ser maior que o da data máxima, se for dá mensagem de erro.

```
190
191   # algumas validações
192
193   if [[ $flagSort == 1 && $flagReverse == 0 && $flagC == 0 && $flagU == 0  && $flagS == 0 && $flagE == 0 ]]; then
194       order=" -rn "
195   fi
196
197   if [[ $flagSort == 1 && $flagReverse == 1 && $flagC == 0 && $flagU == 0  && $flagS == 0 && $flagE == 0 ]]; then
198       order=" -n "
199   fi
200
201   if [[ $flagP == 1 && $flagReverse == 1 && $flagC == 0 && $flagU == 0  && $flagS == 0 && $flagE == 0 ]]; then
202       order=" -rn "
203   fi
204
205   if [[ $flagS == 1 && $flagE == 1 ]]; then
206       if [[ "$dateSecondsS" -gt "$dateSecondsE" ]]; then  # verificar se data minima é menor q a data maxima
207           printf "\n  Data mínima (-s <OPTION>) não pode ser maior que a data máxima (-e <OPTION>)!"
208           message
209           exit 1
210       fi
211   fi
212
```

Temos por fim esta parte do código onde vai ocorrer o *print* da tabela após a execução do script e onde encontramos ainda mais validações para algumas exceções. Começa-se por verificar se a **flagP** está ativa, ou seja, se alguma das opções postas pelo utilizador foi *-p*, se sim, a primeira verificação que se faz é verificar se o número de processos a mostrar usado em *-p* é maior do que o número de processos existentes no computador, se for o caso, dá mensagem de erro e fecha o programa. Caso o argumento de *-p* seja menor ou igual ao número de processos existentes continuamos o código até chegarmos a mais uma verificação. Neste *elif* na linha 321 vamos fazer mais verificações antes de proceder a mostrar a tabela. Verifica se é também utilizado alguma das opções entre *-s* e *-e* (data mínima e máxima). Assumindo que pelo menos uma delas está ativa temos o problema em que sabemos que opções como *-p*, -s, *-e* e -c podem (ou não) fazer diminuir o número de processos que são exibidos na tabela.
Se o utilizador definir um intervalo de datas com com *-s* e *-e* e, o programa devolver 5 processos nesse intervalo de tempo, se quiser mostrar mais processos do que aqueles que retornam de *-s* e/ou *-e* dá erro. Semelhante a isto a linha 327 verifica se a opção *-c* está ativa, assim se o utilizador usar a opção *-c* e esta lhe retornar 5 processos, a verificação seguinte invalida o utilizador de escrever por exemplo "-p 10 -c

<OPTION>" pois se o fizer irá receber uma mensagem de erro pois o número de processos que quer mostrar é maior que o número de processos que retornam do uso da opção *-c*.

Isto é possível através do contadores criados anteriormente, que ao guardar o número de processos no uso de uma certa opção, podemos verificar se o número de processos que queremos exibir é maior ou não. Se não se verificar nenhum erro vai para a linha 340 onde dá print da tabela com as opções selecionadas, **${procs[@]}**" para dar print de todos os valores dentro do array por colunas, **${sort}** para pegar na coluna certa, **${order}** que vai buscar se o valor atual de ordenação é -n ou -rn (dependendo das opções do utilizador) e para mostrar só os processos de *-p* **head -${p}**.

```
316  if [[ $flagP == 1 ]]; then  # se houver '-p'
317      if [[  $i -lt $p ]]; then
318          printf "\n  ERRO: Impossível mostrar %d processos com as opções selecionadas!" "$p"
319          message; exit 1
320
321      elif [[ $flagE == 1 ]] || [[ $flagS == 1 ]]; then           # erro se houver mais processos pedidos em -p q retornados de -s e/ou -e
322          if [[ $se -lt $p ]]; then
323              printf "\n  ERRO: Impossível mostrar %d processos com as opções selecionadas!" "$p"
324              message; exit 1
325          fi
326
327      elif [[ $flagC == 1 ]]; then    # erro se houver mais processos pedidos em -p q retornados de -c
328          if  [[ $cc -lt $p ]]; then
329              printf "\n  ERRO: Impossível mostrar %d processos com as opções selecionadas!" "$p"
330              message; exit 1
331          fi
332      fi
333      printf "%-36s %-16s %12s %12s %11s %16s %16s %12s %12s %15s \n" "COMM" "USER" "PID" "MEM" "RSS" "READB" "WRITEB" "RATER" "RATEW" "DATE"
334      printf "%-36s %-16s %12d %12d %11d %16d %16d %12s %12s     %01s %01s %01s \n" "${procs[@]}" | ${sort} ${order} | head -${p}
335
336  else
337      printf "%-36s %-16s %12s %12s %11s %16s %16s %12s %12s %15s \n" "COMM" "USER" "PID" "MEM" "RSS" "READB" "WRITEB" "RATER" "RATEW" "DATE"
338      if [[ $OPTIND == 1 ]]; then     # se houver só 1 argumento
339          order=" -n "
340          printf "%-36s %-16s %12d %12d %11d %16d %16d %12s %12s     %01s %01s %01s \n" "${procs[@]}" | ${sort} ${order}
341
342      elif [[ ! ${#procs[@]} == 0 ]]; then
343          printf "%-36s %-16s %12d %12d %11d %16d %16d %12s %12s     %01s %01s %01s \n" "${procs[@]}" | ${sort} ${order}
344      else
345          if [[ $flagU == 1 ]]; then
346              printf "\n  ERRO: Utilizador '%s' não existe!" "${u}"
347              message
348              exit 1
349          else
350              printf "\n  ERRO: Não existem processos com os parâmetros especificados!" "${u}"
351              message
352              exit 1
353          fi
354      fi
355  fi
```

De seguida, o *else* da linha 336 serve para agora apenas mostrar os processos quando **flagP=0**, ou seja, não está ativa.

Usando a comparação ***if* $OPTIND == 1**, só vai entrar no if e dar *print* da tabela quando só é passado um argumento, ou seja, o número de segundos, pois este é o único parâmetro obrigatório. Se for passado só o número de segundos, então mostra a tabela por ordem alfabética.

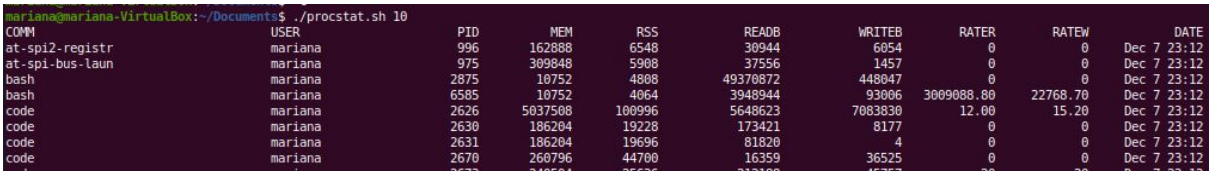
Se for passado mais que um argumento e for verificado que o array *procs* não está vazio então vai dar print usando **"${procs[@]}"**, **${sort}** e **${order}**, como anteriormente.

Caso final se nenhuma dos dois *if* e *elif* se verifica vamos para o último *else*, que só é usado em caso de algo ter sido feito de forma errada. O 1º *if* verifica se a opção *-u* foi utilizada e se sim, se está neste momento a ler o *else* é porque não foi verificado

nenhum **USER** válido, ou seja, não pode dar print a tabela nenhuma e dá mensagem de erro. Caso *-u* não esteja a ser utilizado o último *else* dá erro quando ao usar parâmetros para procurar processos ou datas em específico não foi encontrado nenhum que satisfaz as condições do utilizador, avisando-o disso através da mensagem de erro e fechando o programa.

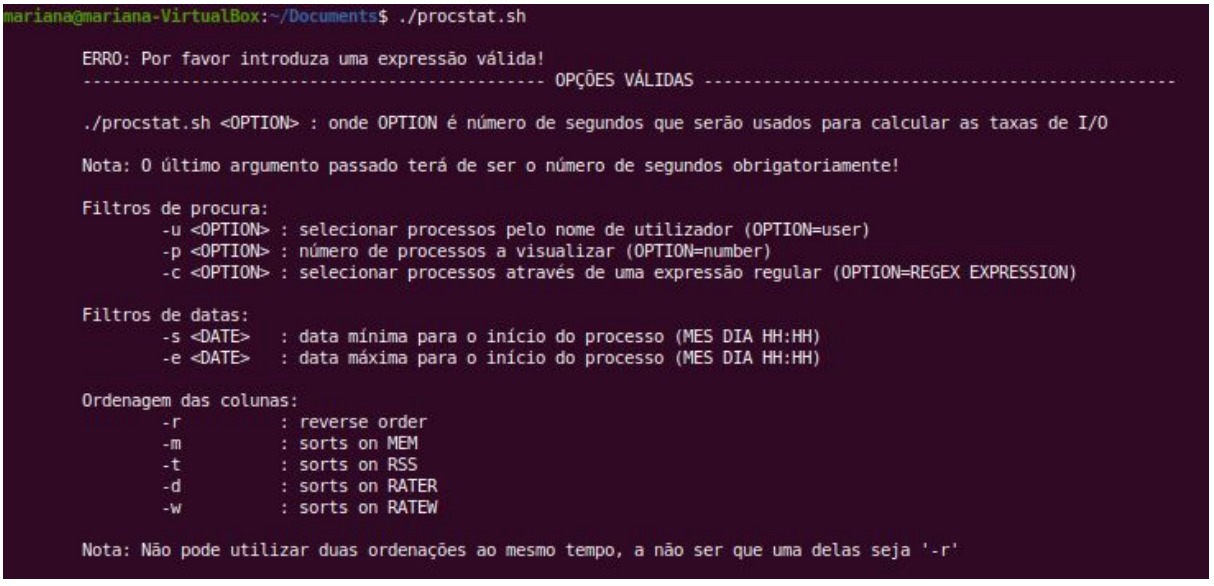
Testes realizados

Teste 1 - Apenas com o número de segundos:



```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh 10
COMM                 USER          PID        MEM        RSS      READB      WRITEB      RATER      RATEW             DATE
at-spi2-registr      mariana       996     162888       6548      30944       6054          0          0     Dec 7 23:12
at-spi-bus-laun      mariana       975     309848       5908      37556       1457          0          0     Dec 7 23:12
bash                 mariana      2875      10752       4808   49370872     448047          0          0     Dec 7 23:12
bash                 mariana      6585      10752       4064    3948944      93006    3009088.80   22768.70     Dec 7 23:12
code                 mariana      2626    5037508     100996    5648623    7083830      12.00      15.20     Dec 7 23:12
code                 mariana      2630     186204      19228     173421       8177          0          0     Dec 7 23:12
code                 mariana      2631     186204      19696      81820          4          0          0     Dec 7 23:12
code                 mariana      2670     260796      44700      16359      36525          0          0     Dec 7 23:12
```

Teste 2 - Sem o número de segundos:



```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh

        ERRO: Por favor introduza uma expressão válida!
        ------------------------------------------------ OPÇÕES VÁLIDAS ------------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'
```

Teste 3 - Utilizar a opção de um utilizador conhecido (*-u*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -u mariana 10
COMM                 USER          PID        MEM        RSS         READB        WRITEB      RATER      RATEW          DATE
at-spi2-registr      mariana       996     162888       6548         30944         25998          0      12.00    Dec 7 23:39
at-spi-bus-laun      mariana       975     309848       5908         37556          1457          0          0    Dec 7 23:39
bash                 mariana      2875      10752       4812     616786133       5900394          0          0    Dec 7 23:39
bash                 mariana     51204      10752       4068       3949162         93132    2990919.90   22788.40    Dec 7 23:39
code                 mariana      2626    5037508      95656       5848829       7242079      12.00      15.20    Dec 7 23:39
```

Teste 4 - Utilizar a opção de um utilizador desconhecido (*-u*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -u naoExisto 10
COMM                 USER          PID        MEM        RSS         READB        WRITEB      RATER      RATEW          DAT

     ERRO: User 'naoExisto'não existe!

        ------------------------------------------- OPÇÕES VÁLIDAS -------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'

mariana@mariana-VirtualBox:~/Documents$
```

Teste 5 - Utilizar a opção de um utilizador conhecido conjuntamente com datas (*-u, -s e -e*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -u mariana -s "Dec 07 23:03" -e "Dec 07 23:10" 10
COMM                 USER          PID        MEM        RSS         READB        WRITEB      RATER      RATEW          DATE
code                 mariana      2727    4572032      61676       6781183       1983816          0          0    Dec 7 23:10
firefox              mariana      2324    3323980     230076      74173477      27340841     223.90     224.20    Dec 7 23:10
gvfsd-trash          mariana      1158     317560       6788         84835          2864          0          0    Dec 7 23:03
systemd              mariana       686      19224       6464      32696208       4714280          0          0    Dec 7 23:03
```

Teste 6 - Utilizar a opção de "start" e "end" date (*-s e -e*) :

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -s "Dec 07 23:10" -e "Dec 07 23:18" 10
COMM                 USER          PID        MEM        RSS         READB        WRITEB      RATER      RATEW          DATE
este print
at-spi2-registr      mariana       996     162888       6548         30944         13830          0          0    Dec 7 23:12
at-spi-bus-laun      mariana       975     309848       5908         37556          1457          0          0    Dec 7 23:12
bash                 mariana      2875      10752       4808     206445516       2031491          0          0    Dec 7 23:12
code                 mariana      2626    5037508     103168       5652583       7091570      12.00      15.20    Dec 7 23:12
code                 mariana      2630     186204      19228        173421          8177          0          0    Dec 7 23:12
code                 mariana      2631     186204      19696         81820             4          0          0    Dec 7 23:12
code                 mariana      2670     259900      45240         16611         46889          0          0    Dec 7 23:12
code                 mariana      2673     240504      27884        227239         48645          0          0    Dec 7 23:12
code                 mariana      2691   15194228     148800      33520491       2147440    1282.90      26.00    Dec 7 23:10
```

Teste 7 - Utilizar apenas da opção de "start" date (*-s*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -s "Dec 07 23:12" 5
COMM                 USER          PID        MEM        RSS         READB        WRITEB      RATER      RATEW          DATE
este print
at-spi2-registr      mariana       996     162888       6548         30944         16182          0          0    Dec 7 23:12
at-spi-bus-laun      mariana       975     309848       5908         37556          1457          0          0    Dec 7 23:12
bash                 mariana     25231      10752       4068       2848459         66902    4342326.60   38640.80    Dec 7 23:27
bash                 mariana      2875      10752       4812     278763488       2729302          0          0    Dec 7 23:12
code                 mariana      2626    5037508     103188       5653522       7092997      24.00      30.40    Dec 7 23:12
code                 mariana      2630     186204      19228        173421          8177          0          0    Dec 7 23:12
```

Teste 8 - Utilizar a opção de "start" e "end" date com uma data de começo superior à final (*-s e -e*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -s "Dec 07 23:30" -e "Dec 07 23:18" 10

        Data mínima (-s <OPTION>) não pode ser maior que a data máxima (-e <OPTION>)!
        ------------------------------------------- OPÇÕES VÁLIDAS -------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!
```

Teste 9- Utilizar a opção de "start" e "end" date com uma data inválida (*-s e -e*)

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh -s "Dec 07 10:15" -e "Dec 32 00:00" 5

        ERRO: Por favor escreva a data no formato 'MÊS DIA HORA' (ex: Jan 01 00:00)!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'
```

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh -s "Dec 07 10:15" -e "Dec 12 25:00" 5

        ERRO: Por favor escreva a data no formato 'MÊS DIA HORA' (ex: Jan 01 00:00)!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'
```

Teste 10 - Ordenar a memória (**MEM**) por ordem *default*, decrescente (*-m*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -m 5
COMM                            USER                 PID         MEM         RSS        READB        WRITEB       RATER       RATEW            DATE
code                            mariana             2691    15201460      157100     35274593      2254536     1686.80       10.60    Dec 7 23:10
Web Content                     mariana             2419     8992200      248164      3420312       310052      209.80      209.80    Dec 7 23:39
code                            mariana             2626     5037508       95668      5850418      7244316           0           0    Dec 7 23:39
code                            mariana             2727     4572032       61552      6786755      1983816           0           0    Dec 7 23:10
code                            mariana             2738     4545168       74384     17974403        64029        2.60        5.20    Dec 7 23:39
gnome-shell                     mariana             1050     3656252      166756      9321844      2772393       56.00      632.00    Dec 7 23:39
firefox                         mariana             2324     3321932      215848     74420865     27435647      287.00      287.60    Dec 7 23:10
pulseaudio                      mariana              705     2793076       10448      2394434        51686     1640.80       40.80    Dec 7 23:39
```

Teste 11 - Ordenar a memória (**MEM**) por ordem crescente (*-m e -r*) :

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -m -r  5
COMM                            USER                  PID         MEM         RSS         READB         WRITEB         RATER         RATEW             DATE
dbus-daemon                     mariana               980        7216        3608         41138          1970             0             0    Dec 7 23:39
dbus-daemon                     mariana               716        9004        4316        122566          5861             0             0    Dec 7 23:39
bash                            mariana              2875       10752        4812     839049364       8038875             0             0    Dec 7 23:39
bash                            mariana             68773       10752        4068       4019242         94814    6052590.20      45868.80    Dec 7 23:45
systemd                         mariana               686       19224        6464      32696208       4714280             0             0    Dec 7 23:03
gnome-session-c                 mariana              1027       90044        3228         24445            64             0             0    Dec 7 23:39
dconf-service                   mariana              1020      156224        4916         30613           649             0             0    Dec 7 23:39
```

Teste 12 - Tentar introduzir dois tipos de ordenação (por exemplo *-m e -t*):

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -m -t  5

        ERRO: Não se pode selecionar mais que 1 tipo de ordenação!
        ------------------------------------------------ OPÇÕES VÁLIDAS ------------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'
```

Teste 13 - Utilizar a opção de filtrar apenas X processos (neste exemplo iremos considerar 3 processos):

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh -p 3 5
COMM                            USER                  PID         MEM         RSS         READB         WRITEB         RATER         RATEW             DATE
at-spi2-registr                 dxogo               41495      162828        7276         38133         720578             0             0    Dec 8 01:19
at-spi-bus-laun                 dxogo               41370      309836        9000         37414           1269             0             0    Dec 8 01:19
bamfdaemon                      dxogo               43219      353572       34600       2118052        2623516             0             0    Dec 8 01:19
```

Teste 14 - Filtrar todos os processos que começam com a letra “d” (*-c “^d.*”*) :

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -c "^d.*"   5
COMM                            USER                  PID         MEM         RSS         READB         WRITEB         RATER         RATEW             DATE
dconf-service                   mariana              1020      156224        4908         30765           953             0             0    Dec 7 23:53
```

Teste 15 - Como verificamos no teste 14, neste momento só existe um processo iniciado pela letra “d”, se pedíssemos ao programa para mostrar 3 tem de dar erro:

```
^Cmariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -c "^d.*" -p 3  5

        ERRO: Impossível mostrar 3 processos com as opções selecionadas!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
```

Teste 16 - Utilizar uma opção que requer um argumento e não associar nenhum número de segundos associado:

```
mariana@mariana-VirtualBox:~/Documents$ ./procstat.sh -p 5

        ERRO: Por favor introduza uma expressão válida!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------

        ./procstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de I/O

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

        Filtros de procura:
                -u <OPTION> : selecionar processos pelo nome de utilizador (OPTION=user)
                -p <OPTION> : número de processos a visualizar (OPTION=number)
                -c <OPTION> : selecionar processos através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de datas:
                -s <DATE>   : data mínima para o início do processo (MES DIA HH:HH)
                -e <DATE>   : data máxima para o início do processo (MES DIA HH:HH)

        Ordenagem das colunas:
                -r          : reverse order
                -m          : sorts on MEM
                -t          : sorts on RSS
                -d          : sorts on RATER
                -w          : sorts on RATEW

        Nota: Não pode utilizar duas ordenações ao mesmo tempo, a não ser que uma delas seja '-r'
```

Teste 17 - Utilização de argumentos inválidos

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh 1 1 1 1 1

        ERRO: Por favor introduza uma expressão válida!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------
```

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh -p 5 5 5 -t 2 -r 1

        ERRO: Por favor introduza uma expressão válida!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------
```

```
dxogo@dx:~/Desktop/SO/TP1$ ./procstat.sh -p 5 a

        ERRO: Por favor introduza uma expressão válida!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------
```

Resultados finais

Posto a execução de todas as linhas de código ditas, dos testes realizados e das validações, podemos ver que de facto, cumprimos com o que nos foi pedido, um *script* onde é possível verificar estatísticas dos diferentes processos em execução no nosso computador com algumas opções.

Conclusão

Ao longo deste trabalho, a nível teórico, consolidamos os nossos conhecimentos sobre os processos, o que são e como funcionam, a sua utilidade e diversidade. A nível prática melhoramos o nosso conhecimento sobre a linguagem Bash, aprimorando as nossas habilidades de programação ao implementar novas metodologias de trabalho e pesquisa.

Em suma, chegamos a uma conclusão pessoal onde a combinação de escrever um *script* e escrever um relatório com base na criação desse código permite-nos otimizar o código e ter, ainda, um maior entendimento sobre o que estamos a criar, adquirindo ainda conhecimentos novos sobre diversos assuntos relacionados com a linguagem bash.

Foi um trabalho cheio de advertências pelo caminho, onde foram precisas muitas horas de discussão entre nós para encontrar soluções a problemas. No entanto, ao chegar ao fim sentimos um grande sentimento de prazer e orgulho de um projeto que nos surpreendeu aos dois.

Fontes

Para a concretização deste trabalho, para além das nossas bases adquiridas em semestres transatos, também nos baseamos na matéria lecionada durante as aulas teóricas e práticas da unidade curricular Sistemas Operativos, nomeadamente na documentação fornecida sobre os processos e nas aulas práticas sobre linguagem *Bash*.

Ao desenvolver o código para a elaboração das respostas às questões propostas, consultamos diversos *sites online* quando nos ocorreu alguma dúvida a respeito do funcionamento de determinados métodos ou funções e os ficheiros para suporte disponibilizados pelos docentes. Em último caso recorremos ao docente da disciplina, Nuno Lau, para esclarecimento de dúvidas cruciais durante o processo de construção do código.