Sistemas Operativos

Estatística de Processamento em Bash

Professor:

Nuno Lau (nunolau@ua.pt)

Pedro Sobral, 98491, P3

Daniel Figueiredo, 98498, P5

As distribuições de percentagem são iguais para os membros do grupo

07/12/2020

Índice

1 - Introdução

No contexto da disciplina de Sistemas Operativos, foi-nos apresentado a realização deste trabalho prático, que consiste na visualização e no tratamento dos processos que estão a *correr* na nossa máquina. O trabalho prático consiste em programar um script *procstat.sh*, de modo a que este permita visualizar a quantidade de memória total, de memória residente em memória física, o número total de bytes I/O, e a taxa de leitura/escrita (bytes/sec) dos processos selecionados nos últimos *sec* segundos.

A implementação do código bash será feita através do IDE *VSCode*, pois é um editor com que ambos os membros grupo já estão familiarizados derivados a unidades curriculares e projetos passados. A execução de todo o trabalho foi suportada através de um repositório no GitHub[1], o que facilita em muito todo o *workflow* da realização do mesmo.

O *script procstat.sh* será feito com base na matéria dada nas aulas práticas, onde foram abordados os conceitos base de programação em bash, que nos permitem listar os processos que se encontram a *correr* no nosso computador. De acordo com o guião teremos de executar vários comandos que irão permitir fazer a filtragem da informação. A nossa ideia inicial para guardar a informação passará por utilizar arrays associativos, visto são estruturas de dados onde é possível identificar cada elemento por uma *key*, e por um *value*, de um modo geral é um pouco como os *HashMaps* em *Java*, por exemplo. Depois de ter a informação dos processos toda tratada/formatada e devidamente guardada, iremos criar as *expressões condicionais* para tratar a informação de acordo com as opções passadas no terminal, quando se corre o programa.

Com a realização deste trabalho prático, esperamos veemente alargar os nossos horizontes no que toca a programar em bash, e perceber um pouco mais do funcionamento dos processos do sistema operativo (*Linux - Ubuntu*), visto que é bastante importante para esta unidade curricular.

[1] – O repositório encontra-se privado, pelo menos até ao dia da entrega do trabalho (07/12/2020)

2 - Estrutura do Código

2.1 – Declaração de variáveis globais

Primeiramente, para o tratamento da informação, pensamos utilizar *arrays* associativos, pois são estruturas de dados que permitem referenciar vários dados por um único nome, deste modo, para nos referirmos a um determinado valor utilizamos uma "chave", que está diretamente ligada a um valor. Assim achamos por bem, criar este tipo de estruturas de dados e inicializamos os *arrays arrayAss, argOpt, R1* e *W1* (Fig.1). Também criamos as variáveis *i* e *re*, que irão servir para a validação de argumentos.

```
#Arrays
declare -A arrayAss=() # Array Associativo: está guardado a informação de cada processo, sendo a 'key' o PID
declare -A argOpt=()   # Array Associativo: está guardada a informação das opções passadas como argumentos na chamada da função
declare -A R1=()       # Array Associativo: está guardada a informação do rchar1
declare -A W1=()       # Array Associativo: está guardada a informação do wchar1

i=0                    # Inicialização da variável i, usada na condição de verificação de opçoes de ordenac
re='^[0-9]+([.][0-9]+)?$' # Expressão regex
```

Figura 1- Inicialização de variáveis globais.

- ***arrayAss*** – *array* associativo que contém informações acerca de todos os processos que, posteriormente tenham passado pelos processos de validação no nosso código, tem como *key* o *PID* de cada processo, não havendo, assim, colisão de informação.

- ***argOpt*** - *array* associativo onde estão guardadas as informações das opções passadas como argumento, quando se corre o programa, sendo a *key*, a opção passada pelo utilizador e o seu *value* o argumento da mesma.

- **R1** – *array* associativo onde está guardada a informação sobre o número total de bytes de I/O a ler.

- **W1** – *array* associativo onde está guardada a informação sobre o número total de bytes de I/O a escrever.

- **i** – Inicialização da variável i com valor zero, que vai ser usada, posteriormente, na validação das opções de ordenação.

- **re –** Expressão *regex*, usada na validação das opções com argumento, para verificar se o argumento passado é um número(inteiro ou *float*).

2.2 – Tratamento de opções

Para o tratamento de opções, utilizamos o comando *getopts*, que nos permite passar várias opções. Caso essas opções sejam válidas, então o comando *getopts*, vai fazer o tratamento destas e fazer as tarefas desejadas. Caso as opções passadas pelo utilizador não sejam válidas, então vamos chamar a função *opcoes*, que vai apresentar ao utilizador uma lista de opções válidas para correr o programa, terminando assim o programa.

2.2.1 – Função opcoes()

```
#Função para quando argumentos passados são inválidos
function opcoes() {
    echo "*********************************************************************************************"
    echo "OPÇÃO INVÁLIDA!"
    echo "    -c          : Seleção de processos a utilizar através de uma expressão regular"
    echo "    -u          : Seleção de processos a visualizar através do nome do utilizador"
    echo "    -r          : Ordenação reversa"
    echo "    -s          : Seleção de processos a visualizar num periodo temporal - data mínima"
    echo "    -e          : Seleção de processos a visualizar num periodo temporal - data máxima"
    echo "    -d          : Ordenação da tabela por RATER (decrescente)"
    echo "    -m          : Ordenação da tabela por MEM (decrescente)"
    echo "    -t          : Ordenação da tabela por RSS (decrescente)"
    echo "    -w          : Ordenação da tabela pOR RATEW (decrescente)"
    echo "    -p          : Número de processos a visualizar"
    echo "   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo"
    echo "Último argumento: O último argumento passado tem de ser um número"
    echo "*********************************************************************************************"
}
```

Figura 2 - Função opcoes()

Nesta lista que é impressa no terminal do utilizador, podemos observar as opções válidas, sendo elas: *-c, -u, -r, -s, -e, -d, -m, -t, -w, -p:*

- A opção *-c*, vai permitir filtrar os processos através da expressão *regex* que o utilizador passar como argumento desta opção, aparecendo apenas, os processos que estejam conforme a expressão passada.
- A opção *-u*, permite que, ao passar como argumento o nome de um utilizador, apenas apareçam os processos que estejam a ser feitos por esse mesmo utilizador.
- A opção *-r, -d, -m, -t, -w*, servem apenas para ordenar os processos, sendo os processos de ordenação por ordem reversa, ordenação por *RATER* (decrescente), ordenação por *MEM* (decrescente), ordenação por *RSS* (decrescente) e ordenação por *RATEW* (decrescente), respetivamente. Podendo intercalar a opção *-r* com as demais opções.
- A opção *-s* e *-e*, permitem-nos filtrar os processos por data inicial e data final, passando o utilizador como argumento de *-s* a data que pretende que os processos tenham iniciado e para o argumento *-e* a data para que pretende que os processos tenham terminado (podendo ser usado separadamente).
- A opção *-p*, serve para que ao utilizador apenas lhe apareça a quantidade de processos que este desejar. Ou seja, supondo que o utilizador passou como argumento o número 10, apenas lhe irão aparecer 10 processos.

2.2.2 - *getopts*

Como referido anteriormente, utilizamos a comando *getopts* (Fig.3) que nos vai permitir passar várias opções e fazer o tratamento das mesmas.

```
44  while getopts "c:u:rs:e:dmtwp:" option; do
```

Figura 3 - getopts e as suas opções

Como podemos observar na (Fig. 3) ao comando *getopts* foram passadas várias opções e algumas destas têm ":" à frente delas e porquê? Porque, como lecionado, nas aulas de "Sistemas Operativos", as opções que possuem ":" à frente delas é porque vão necessitar de um argumento passado para elas.

```
45      # Verificação se o último argumento passado é um numero
46      if ! [[ ${@: -1} =~ $re ]]; then
47          opcoes
48          exit
49      fi
```

Figura 4 - Verificação do último argumento

Dentro do comando *getopts*, fizemos uma condição *if* que nos vai permitir fazer a validação do último argumento passado (que obrigatoriamente tem de ser um número), ou seja, caso o último argumento passado não faça *match* com *re* (Fig. 1) então será chamada a função *opcoes()* (Fig. 2) e o programa irá terminar.

```
51      #Adicionar ao array argOpt as opcoes passadas ao correr o procstat.sh, caso existam adiciona as que são passadas, caso não, adiciona "nada"
52      if [[ -z "$OPTARG" ]]; then
53          argOpt[$option]="nada"
54      else
55          argOpt[$option]=${OPTARG}
56      fi
```

Figura 5 – Adição das opções ao array argOpt

Ainda dentro deste comando, iremos então prosseguir à adição das opções passadas e dos seus respetivos argumentos. Caso a opção passada não possua nenhum argumento, a *key* que vai ficar no array *argOpt* vai ser essa mesma opção, e caso não tenha nenhum argumento então vai ficar com o seu *value* igual a "nada". Por outro lado, se a opção passada tivesse argumento, a *key* que vai ficar no array *argOpt* vai ser essa mesma opção, contudo, o seu *value*, desta vez, vai ser o argumento passado com a opção.

```
60      case $option in
61      c) #Seleção de processos a utilizar atraves de uma expressão regular
62          str=${argOpt['c']}
63          if [[ $str == 'nada' || ${str:0:1} == "-" || $str =~ $re ]]; then
64              echo "Argumento de '-c' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada." >&2
65              exit 1
66          fi
67          ;;
68      s) #Seleção de processos a visualizar num periodo temporal - data mínima
69          str=${argOpt['s']}
70          if [[ $str == 'nada' || ${str:0:1} == "-" || $str =~ $re ]]; then
71              echo "Argumento de '-s' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada." >&2
72              exit 1
73          fi
74          ;;
75      e) #Seleção de processos a visualizar num periodo temporal - data máxima
76          str=${argOpt['e']}
77          if [[ $str == 'nada' || ${str:0:1} == "-" || $str =~ $re ]]; then
78              echo "Argumento de '-e' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada." >&2
79              exit 1
80          fi
81          ;;
82      u) #Seleção de processos a visualizar através do nome do utilizador
83          str=${argOpt['u']}
84          if [[ $str == 'nada' || ${str:0:1} == "-" || $str =~ $re ]]; then
85              echo "Argumento de '-u' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada." >&2
86              exit 1
87          fi
88          ;;
89      p) #Número de processos a visualizar
90          if ! [[ ${argOpt['p']} =~ $re ]]; then
91              echo "Argumento de '-p' tem de ser um número ou chamou sem '-' atrás da opção passada." >&2
92              exit 1
93          fi
94          ;;
```

Figura 6 - Tratamento de opções com argumento

Para as opções com argumentos tivemos de fazer um tratamento diferente, comparando com as que não têm argumento e com a opção "-p".

Para cada opção guardamos os respetivos *values* de cada opção em *str*. Em seguida, fizemos condições para poder ver se, caso não tivesse sido dado nenhum argumento então o *value* das opções ficaria com o *value "*nada*"*. Outra condição, para caso a primeira passasse é o facto de o *value* tivesse como primeiro caracter "-", que só acontece caso, por exemplo ao *correr* o programa com as opções -u -c "co.*" 10, o *value* da opção "-u" iria ser "-c" o que não pode acontecer. Caso estas duas condições passassem, só restava ver se o *value* é um número, visto que para estas opções só nos interessam *strings*, logo fizemos a condição de que se *str* fizesse match com *re (*Fig. 1) então o programa teria de terminar.

Para a opção "-p" tem de ser diferente, visto que, esta opção apenas pode aceitar números como argumento. Logo, dentro da condição do *if* fizemos um *match* com *re* e como tem a negação atrás da condição vai significar que o programa só acaba caso o argumento passado não seja um número.

```
95      r) #Ordenação reversa
96
97          ;;
98      m | t | d | w)
99
100         if [[ $i = 1 ]]; then
101             #Quando há mais que 1 argumento de ordenacao
102             opcoes
103             exit
104         else
105             #Se algum argumento for de ordenacao i=1
106             i=1
107         fi
108         ;;
109
110     *) #Passagem de argumentos inválidos
111         opcoes
112         exit
113         ;;
114     esac
115
116 done
```

Figura 7 - Tratamento das opções sem argumento

Para as opções que não precisam de argumentos passados, fizemos apenas a verificação de o programa não poder ser *corrido* caso haja associação entre as opções "-m", "-t", "-d" e "-w", podendo haver associação com "-r", visto que esta opção faz apenas a ordenação reversa a estas opções.

Para estas opções usamos a variável global *i (*Fig. 1*)* que foi inicializada com valor zero. Supondo que o programa foi chamado com as opções -m -r -t 10, o *getopts* ao receber a opção "-m" vai ver se o valor de *i* é igual a um e caso não seja vai meter *i* igual a um, na segunda opção vai iterar com a opção "–r" que é uma opção válida para usar com "-m", o *getopts* não vai fazer nada na opção "-r" visto que esta não tem nada para fazer, em seguida vai à opção "-t" e vai ver se tem o valor de *i* igual a um, como se verifica esta última condição então o programa vai chamar a função *opcoes()* e vai terminar o programa.

2.2.3 – Número mínimo de argumentos

```
120 if [[ $# == 0 ]]; then
121     echo "Tem de passar no mínimo um argumento (segundos)."
122     opcoes
123     exit 1
124 fi
```

Figura 8 - Número mínimo de argumentos

Para que o programa *corra* corretamente é necessário que lhe seja passado um argumento.

Neste trecho de código, começamos por fazer uma condição *if*, que vai verificar se foi passado no mínimo um argumento, podendo este ser uma *string* ou um número (apesar de ser outro erro caso o argumento passado seja uma *string* como iremos ver posteriormente).

2.2.4 – Último argumento

```
126 # Verificação se o último argumento passado é um numero
127
128 if ! [[ ${@: -1} =~ $re ]]; then
129     echo "Último argumento tem de ser um número."
130     opcoes
131     exit 1
132 fi
```

Figura 9 - Verificar se último argumento é um número

Regra geral do código, o último argumento terá sempre de ser um número, caso seja uma *string* ou qualquer outro caractere que não seja um número, o programa vai terminar devido à condição *if*, que vai tentar fazer *match* com a variável *re* (Fig. 1) e caso falhe vai imprimir no terminal a frase que está em *echo*, vai chamar a função *opcoes* e vai terminar o programa.

3 – Listar os Processos

3.1 – Primeira Leitura de *rchar, e wchar*

A função *listarProcessos* (Fig. 10) é uma das principais funções do *script*, sendo que é aqui que onde se leem as informações dos processos e se trata desses dados de forma eficiente.

```
118 #Tratamento dos dados lidos
119 function listarProcessos() {
120
121     #Cabeçalho
122     printf "%-30s %-16s %15s %12s %12s %12s %12s %12s %12s %16s\n" "COMM" "USER" "PID" "MEM" "RSS" "READB" "WRITEB" "RATER" "RATEW" "DATE"
123     for entry in /proc/[[:digit:]]*; do
124         if [[ -r $entry/status && -r $entry/io ]]; then
125             PID=$(cat $entry/status | grep -w Pid | tr -dc '0-9') # ir buscar o PID
126             rchar1=$(cat $entry/io | grep rchar | tr -dc '0-9')   # rchar inicial
127             wchar1=$(cat $entry/io | grep wchar | tr -dc '0-9')   # wchar inicial
128
129             if [[ $rchar1 == 0 && $wchar == 0 ]]; then           # não conseguir captar informação
130                 continue
131             else
132                 R1[$PID]=$(printf "%12d\n" "$rchar1")
133                 W1[$PID]=$(printf "%12d\n" "$wchar1")
134             fi
135         fi
136
137     done
```

Figura 10 - Função listarProcessos()

Começamos por *printar* o cabeçalho da nossa tabela, visto que este será sempre impresso, e só queremos que este seja *printado* uma vez a cada execução do *script*. Seguidamente iniciamos um ciclo *for*, este ciclo *for*, será somente para realizarmos uma leitura das variáveis *rchar* e *wchar*, que cada processo dispõe, isto acontece por uma questão de eficiência do código, desta maneira conseguimos fazer *sleep* uma vez em toda a execução do *script*. No ciclo *for*, vamos percorrer todos os processos ativos na máquina, em que o nome da diretoria são somente números (Fig.11). De seguida aplicamos uma condição para verificar se temos permissões no *status*, e no *io*, no determinado processo em que o ciclo *for* itera, tendo permissões, guardamos na variável *PID*, o *pid* do processo, utilizamos um *grep*, e o *tr* de modo a que só retornasse os números correspondentes ao *pid*.

```
1       1160    1202    1296    15      16      1966    2049    2234
10      1161    1203    13      151     160283  1968    2051    2237
1042    1164    1206    130     151605  165078  1985    2073    2249
1059    1174    121     1305    151702  165094  1990    21      2253
11      1179    1211    1306    151865  166     1992    2141    2255
1101    1182    122     134     151867  168014  2       2162    2264
114     119     122665  1350    152151  17      20      2167    2273
115     1192    123     136     152349  18      2004    2177    2282
1152    1197    1232    136243  154029  1904    2008    2184    228558
1153    1198    124     139     154634  1945    2015    2198    228563
1155    1199    1251    139088  15494   1953    2021    22      2288
1157    12      1266    14      1557    1954    2025    2227    23
1159    120     127     1461    1580    1960    2030    2231    2303
```

Figura 11 - Exemplos de processos que o ciclo for percorre

Guardamos nas variáveis *rchar1 e wchar1* os valores correspondentes ao rchar e ao wchar, respetivamente, utilizamos um *grep* e um *tr* de modo a só retornar para a função os valores correspondentes ao que queremos.

Prosseguindo no código, encontramos um *if*, este *if* é essencial caso os valores de caso os valores de *rchar1 e wchar1* sejam 0, se porventura isso acontecer utilizamos a palavra reservada *continue*, que faz avançar para o próximo processo no ciclo *for*. Caso as condições não se verifiquem, adicionamos aos arrays associativos *R1*, e *W1*, os valores de *rchar1* e *wchar1*, respetivamente, onde a *key* pela qual guardamos os valores nos arrays é o pid, calculado logo no início deste bloco de código.

3.2 - Sleep

Ao fim de todos os processos serem lidos pelo *for*, este acaba e executamos o comando *sleep (*Fig.12*)*, onde lhe passamos o argumento *$1*, que é o único argumento passado à função *listarProcessos()*.

```
138
139     sleep $1 # tempo em espera
```

Figura 12 - sleep

3.3 – Leitura de estatísticas

Executamos agora outro ciclo *for (*Fig.13*)*, sendo que é neste em que a maioria da informação vai ser extraída, este ciclo itera sobre os processos que estão ativos na nossa máquina, ou seja, itera da mesma maneira que o *for* anterior. De seguida aplicamos uma condição para verificar se temos permissões no *status*, e no *io*, no determinado processo em que o ciclo *for* itera.

Vamos extrair o *pid*, para a variável *PID*, através de um *grep* e de um *tr*, de modo a que só fique guardado na variável o número do mesmo. A seguir, guardamos na variável *user*, o nome do utilizador do processo, sendo este extraído através do *PID*, e do comando *ps*. Extraímos o VmSize, e o VmRss, com recurso ao *grep* e ao *tr*. E por último nesta fase vamos buscar o nome do processo, ficando este guardado na variável *comm*, utilizamos o *tr* " " "_", de modo a que caso algum processo possui-se mais que uma palavra no seu nome, o espaço fica-se substituído por um underline, e desta forma já não iria estragar a formatação da tabela que irá ser criada mais à frente.

```
141     for entry in /proc/[[:digit:]]*; do
142
143         if [[ -r $entry/status && -r $entry/io ]]; then
144
145             PID=$(cat $entry/status | grep -w Pid | tr -dc '0-9') # ir buscar o PID
146             user=$(ps -o user= -p $PID)                          # ir buscar o user do PID
147
148             VmSize=$(cat $entry/status | grep VmSize | tr -dc '0-9') # ir buscar o VmSize
149             VmRss=$(cat $entry/status | grep VmRSS | tr -dc '0-9')   # ir buscar o VmRss
150
151             comm=$(cat $entry/comm | tr " " "_") # ir buscar o comm,e retirar os espaços e substituir por '_' nos comm's com 2 nomes
```

Figura 13 - Continuação função listarProcessos()

Seguidamente e através de expressões condicionais fomos tentar otimizar o nosso código, ou seja, se ao correr o *script*, foram passadas as opções "-c" ou "-u", temos de filtrar os processos correspondentes as essas opções e aos seus respetivos argumentos (Fig.14), de modo a que só sejam adicionados processos que o utilizador ao correr o *script* pediu.

Ora, para a opção "-u", temos um *if*, que verifica se essa opção se encontra no array associativo *argOpt*, que como já foi explicado anteriormente é um array onde ficam guardadas as opções e os argumentos das mesmas passadas ao correr o *script*, e se o *user* do processo em

questão for diferente daquele que foi passado à opção "-u", então passamos para o próximo processo, descartando assim o anterior através do comando *continue*.

Para a opção "-c", o procedimento e o raciocínio é o mesmo utilizado na opção anterior, vamos ver se o *argOpt* tem o "-c" como *key*, e se o *comm* do processo for diferente daquele que foi passado no terminal aquando se correu o *script*, é usado o comando *continue*, descartamos este processo, e passamos para o próximo.

```
153             if [[ -v argOpt[u] && ! ${argOpt['u']} == $user ]]; then
154                 continue
155             fi
156
157             #Seleção de processos a utilizar atraves de uma expressão regular
158             if [[ -v argOpt[c] && ! $comm =~ ${argOpt['c']} ]]; then
159                 continue
160             fi
```

Figura 14 - Seleção de processos, "-u" e "-c"

Prosseguindo no nosso código, chegamos à parte das datas, datas dos processos, e data passadas como argumentos (Fig.15). Primeiramente, passamos o código para que a data fique no formato inglês, de seguida, através do comando *ps* e do *PID*, já calculado anteriormente, guardamos na variável *startDate*, a data de início do processo, a seguir passamos a data para o formato "**mmm dd HH:MM"**, ou seja, mês no diminutivo, dia, hora, e minutos, pois é neste formato que a data é apresentada no guião do trabalho prático. Na linha seguinte de código, passamos para segundos a data de início do processo, pois será útil, para comparações futuras.

```
162             LANG=en_us_8859_1
163             startDate=$(ps -o lstart= -p $PID) # data de início do processo atraves do PID
164             startDate=$(date +"%b %d %H:%M" -d "$startDate")
165             dateSeg=$(date -d "$startDate" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}') # data do processo em segundos
```

Figura 15 - Data processos

Temos também validações do género da (Fig.14), no entanto agora para as opções "-s" e "-e", como já referido anteriormente com estas condições são bastante uteis pois permitem filtrar a informação de acordo com o que foi passado no terminal ao corre o *script*.

Para a opção "-s" (Fig.16), através de um *if*, verificamos se essa opção foi passada, caso tenha sido avança e na variável *start* irá ficar registado em segundos a data passada como argumento da opção "-s", de seguida noutro *if*, fazemos uma comparação, se o número de segundos do processo for menor que o número de segundos passados na opção "-s", usamos o comando *continue*, descartando este processo, e avançando para o próximo.

Para a opção "-e" (Fig.16), a lógica é a mesma, no primeiro *if*, verificamos se essa opção comparece no argOpt, caso se verifique, guarda na varável *end*, em segundos a data passada como argumento na opção "-e". No segundo *if*, comparamos se o número de segundos do processo é maior que o número de segundos passados na variável "-e", caso seja, descartamos este processo, e avançamos para o próximo, através do comando *continue*.

```
169             if [[ -v argOpt[s] ]]; then #Para a opção -s
170                 start=$(date -d "${argOpt['s']}" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}') # data mínima
171 
172                 if [[ "$dateSeg" -lt "$start"  ]]; then
173                     continue
174                 fi
175             fi
176 
177             if [[ -v argOpt[e] ]]; then #Para a opção -e
178                 end=$(date -d "${argOpt['e']}" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}')   # data máxima
179 
180                 if [[ "$dateSeg" -gt "$end" ]]; then
181                     continue
182                 fi
```

Figura 16 - Seleção de processos, "-s" e "-e"

Neste momento vamos voltar a ler o *rchar* e o *wchar*, sendo que ficam guardados nas variáveis *rchar2* e *wchar2*, respetivamente, com a ajuda do *grep* e do *tr*, guardamos assim os valores nas variáveis referidas (Fig.17).

A seguir fomos calcular as variáveis *rateR* e *rateW*. Começando pelo *rateR*, fizemos uma subtração, ficando guardada na variável *subr*, entre a segunda leitura, *rchar2*, e a primeira leitura, que se encontra guardada no array associativo, *R1*, sendo a *key* o *PID*, e posteriormente fazemos a divisão pelo número de segundos que o comando *sleep* esteve ativo, *$1*, para realizar esta operações utilizamos o comando *bc* e usamos ainda o *scale=2*, de modo a que a divisão fique com 2 casas decimais. Quanto ao rateW, o processo é semelhante, realizamos uma subtração, guardada na variável *subw*, entre o *wchar2* e a primeira leitura que se encontra no array associativo *W1*, sendo a *key* o *PID*, assim fazemos a subtração e de seguida a divisão pelo $1, sendo estas operações também realizadas com o comando *bc*, utilizamos também o *scale=2*, para que o resultado seja calculado com 2 casas decimais.

```
186             rchar2=$(cat $entry/io | grep rchar | tr -dc '0-9') # rchar apos s segundos
187             wchar2=$(cat $entry/io | grep wchar | tr -dc '0-9') # wchar apos s segundos
188             subr=$rchar2-${R1[$PID]}
189             subw=$wchar2-${W1[$PID]}
190             rateR=$(echo "scale=2; $subr/$1" | bc -l)       # calculo do rateR
191             rateW=$(echo "scale=2; $subw/$1" | bc -l)       # calculo do rateW
```

Figura 17 - Segunda leitura rchar e wchar e rates

Chegamos a um ponto onde já temos todas as informações para que seja possível construir a tabela, decidimos criar o array associativo *arrayAss*, sendo neste array que irá ser guarda toda a informação recolhida até agora e assim formar a tabela dos processos (Fig.18). Para guardarmos a informação já formatada no array, utilizamos o comando *printf*, sendo que deste modo exequível guardar os tipos de valores que cada variável tem, decimal, float, string.

```
arrayAss[$PID]=$(printf "%-27s %-16s %15d %12d %12d %12d %12d %15s %15s %16s\n" "$comm" "$user" "$PID" "$VmSize" "$VmRss" "${R1[$PID]}" "${W1[$PID]}" "$rateR" "$rateW" "$startDate")
```

Figura 18 - Guardar informações em arrayAss

É aqui que o segundo ciclo *for* implementado, acaba, sendo logo a seguir chamada a função *prints (Fig.19)*, que será explicada na próxima secção deste relatório.

```
194         fi
195     done
196
197     prints
198 }
```

Figura 19 - Chamada da função prints(), e fim da função listar processos

4 – Impressão ordenada dos dados

A impressão ordenada dos dados é feita com a chamada da função *prints()* (Fig.20).

```
195 function prints() {
196
197     if ! [[ -v argOpt[r] ]]; then
198         ordem="-rn"
199     else
200         ordem="-n"
201     fi
202
203     #Nº processos que queremos ver
204     if [[ -v argOpt[p] ]]; then
205         p=${argOpt['p']}
206     #Se não dermos nenhum valor ao -p, fica com o valor do tamanho do array
207     #Ou seja printa todos
208     else
209         p=${#arrayAss[@]}
210     fi
211
212     if [[ -v argOpt[m] ]]; then
213         #Ordenação da tabela pelo MEM
214         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k4 | head -n $p
215
216     elif [[ -v argOpt[t] ]]; then
217         #Ordenação da tabela pelo RSS
218         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k5 | head -n $p
219
220     elif [[ -v argOpt[d] ]]; then
221         #Ordenação da tabela pelo RATER
222         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k8 | head -n $p
223
224     elif [[ -v argOpt[w] ]]; then
225         #Ordenação da tabela pelo RATEW
226         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k9 | head -n $p
227
228     else
229         #Ordenação default da tabela, ordem alfabética dos processos
230         ordem="-n" #Como é por ordem alfabética temos de mudar a ordem para '-n'
231
232         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k1 | head -n $p
233
234     fi
235
236 }
```

Figura 20 - Função prints()

4.1 – Tratamento da ordem e do número de processos

O resultado ordenado dos dados filtrados é baseado na leitura e impressão, com a respetiva ordenação dos dados do array associativo *arrayAss*. No início desta função (Fig.21), é verificado se a opção "-r" foi passada para se *correr* o programa no *argOpt* através de uma expressão condicional com o operador "-v". Caso a opção "-r" não tenha sido usada, com o operador "-v" vai-se verificar que essa opção se não se encontra presente nesse *array*, fazendo assim que se entre no primeiro *if*, criando a variável *ordem* que fica com valor "-rn", valor esse que vai permitir aos *printf's* imprimirem as informações ordenadas de forma crescente. Caso a opção se encontre dentro do *array argOpt* então vai entrar no *else* e *ordem* vai ficar com o valor "-n" imprimindo as informações por ordem decrescente.

De seguida, temos outra condição *if (*Fig.21*)*, que nos vai permitir ver qual é o valor da opção "-p", caso a opção "-p" tenha sido chamada para *correr* o programa então o operador "-v" vai averiguar se esta opção é uma *key* no *array argOpt* e caso seja entra na condição. Entrando na condição, vai ser criada a variável local *p* que fica com o valor de $argOpt{['p']}, ou seja, fica com o argumento que foi passado ao utilizar-se esta opção, para ser usada posteriormente. Caso não entre, a variável *p* vai ter valor igual ao tamanho do *array argOpt*.

```
215 function prints() {
216
217     if ! [[ -v argOpt[r] ]]; then
218         ordem="-rn"
219     else
220         ordem="-n"
221     fi
222
223     #Se não dermos nenhum valor ao -p, fica com o valor do tamanho do array
224     #Ou seja printa todos
225     if ! [[ -v argOpt[p] ]]; then
226         p=${#arrayAss[@]}
227     #Nº de processos que queremos ver
228     else
229         p=${argOpt['p']}
230     fi
231
```

Figura 21 – Validação reverse e opção "-p"

4.2 – *Prints* de acordo com as opções

As verificações seguintes averiguam qual das opções foi passada como opção de ordenação no *argOpt*, de modo a imprimir no terminal os dados que se pretendem (Fig.22).

Caso a opção com que se correu o programa tenha sido a opção "-m" então vai entrar no *if* referente a essa opção (" -v argOpt[m]") e o operador "-v" vai verificar se a opção "-m" é uma das *keys* presentes em argOpt. Através do *printf* vai ser possível imprimir toda a informação que está guardada no *array arrayAss*. Para além do *printf*, também o comando *sort* é importante para imprimir a informação, pois é através deste que se vai ordenar a informação pela opção que queremos, sendo que a opção "-k" designa o local onde queremos operar, daí ser seguida por um número (por exemplo "-k4").

Supondo que a opção passada foi a opção "-m", então, através do *printf* e do comando *sort* vamos poder ordenar a informação pelo VmSize, podendo ordenar esta informação por ordem crescente, caso *ordem* seja igual a "-rn" vai ordenar por ordem decrescente e caso seja "-n" vai

ordenar por ordem crescente. Como a opção passada foi a opção "-m" então vamos usar *sort $ordem -k4*, ordenando assim, o nosso *array arrayAss*, pela quarta coluna da informação.

Outra informação que está presente no *printf* é a informação relativa ao comando *head*, que nos vai permitir imprimir quantos processos o utilizador do programa quiser. Como dito, anteriormente, esta opção vai imprimir todos os processos caso a opção "-p" não tenha sido utilizada. Uma boa utilização para esta opção seria o facto de o utilizador apenas querer imprimir o processo que ocupe mais memória virtual, podia então correr o programa com as opção "-m -p 1 10", ordenando o programa por VmSize e imprimindo apenas um processo devido à opção "-p" que foi chamada com o argumento um, obtendo assim o processo que está a ocupar mais memória física.

```
232     if [[ -v argOpt[m] ]]; then
233         #Ordenação da tabela pelo MEM
234         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k4 | head -n $p
235 
236     elif [[ -v argOpt[t] ]]; then
237         #Ordenação da tabela pelo RSS
238         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k5 | head -n $p
239 
240     elif [[ -v argOpt[d] ]]; then
241         #Ordenação da tabela pelo RATER
242         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k8 | head -n $p
243 
244     elif [[ -v argOpt[w] ]]; then
245         #Ordenação da tabela pelo RATEW
246         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k9 | head -n $p
247 
248     else
249         #Ordenação default da tabela, ordem alfabética dos processos
250         ordem="-n" #Como é por ordem alfabética temos de mudar a ordem para '-n'
251 
252         printf '%s \n' "${arrayAss[@]}" | sort $ordem -k1 | head -n $p
253 
254     fi
255 
256 }
```

Figura 22 – Print da tabela

O script vai terminar com uma linha de código (Fig.23) que vai permitir que se vão buscar todos os valores e se imprimam esses mesmos valores.

```
238 listarProcessos ${@: -1} #este agumento passado, é para os segundos, visto que é passado em todas as opções, e é sempre o último
```

Figura 23 - Chamada da função listarProcessos()

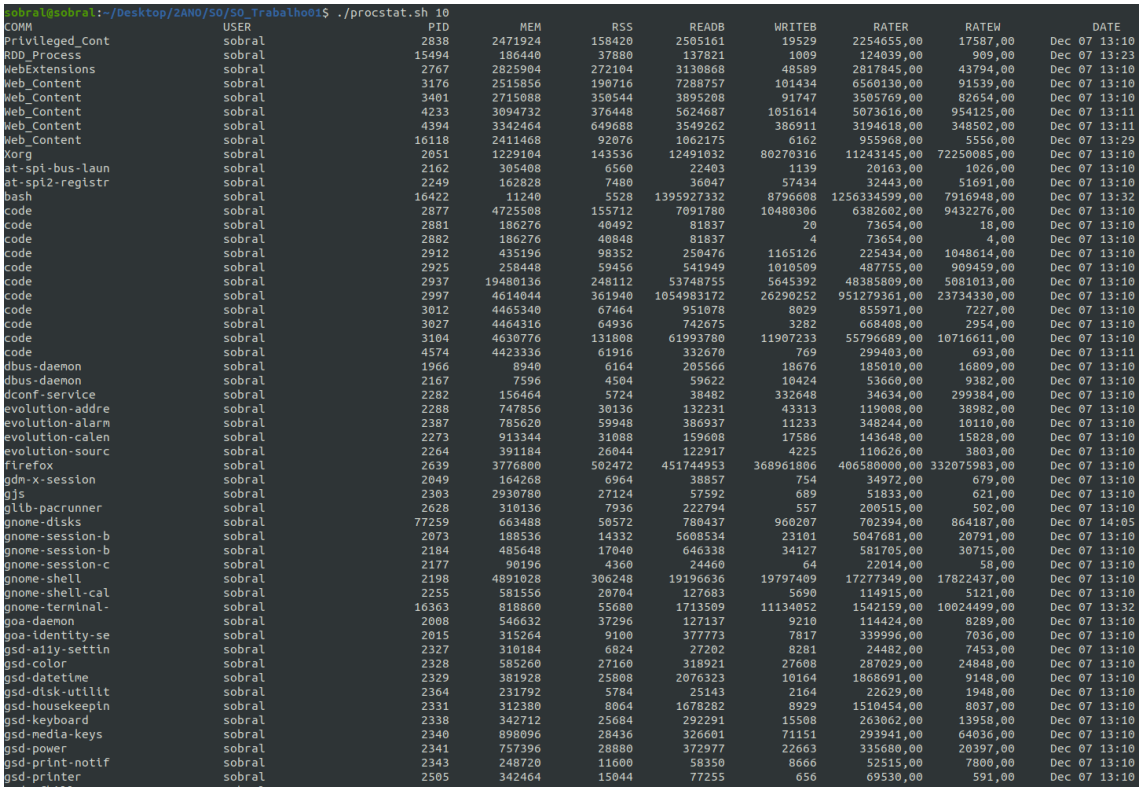
Esta função é chamada com um argumento, este vai ser utilizado para fazer o comando *sleep* e as contas referentes ao *rateR* e ao *rateW* como já referido anteriormente. Este argumento chamado vai ser o último argumento passado ao *correr* o programa, ou seja, vai ser o número de segundos.

5 – Resultados

Nesta parte do relatório, irão ser mostrados algumas das maneiras possíveis para executar o *script procstat.sh*[2].

5.1 – Resultados para argumentos válidos

O primeiro resultado, e também o mais fácil, é executar o *script*, e passar só como argumento o número de segundos em que o *sleep* irá estar ativo (Fig.24).



```
sobral@sobral:~/Desktop/2ANO/SO/SO_Trabalho01$ ./procstat.sh 10
COMM                 USER                 PID         MEM         RSS       READB       WRITEB       RATER       RATEW             DATE
Privileged_Cont      sobral              2838     2471924      158420     2505161        19529     2254655,00    17587,00    Dec 07 13:10
RDD_Process          sobral             15494      186440       37880      137821         1009      124039,00      909,00    Dec 07 13:23
WebExtensions        sobral              2767     2825904      272104     3130868        48589     2817845,00    43794,00    Dec 07 13:10
Web_Content          sobral              3176     2515856      190716     7288757       101434     6560130,00    91539,00    Dec 07 13:10
Web_Content          sobral              3401     2715088      350544     3895208        91747     3505769,00    82654,00    Dec 07 13:10
Web_Content          sobral              4233     3094732      376448     5624687      1051614     5073616,00   954125,00    Dec 07 13:11
Web_Content          sobral              4394     3342464      649688     3549262       386911     3194618,00   348502,00    Dec 07 13:11
Web_Content          sobral             16118     2411468       92076     1062175         6162      955968,00     5556,00    Dec 07 13:29
Xorg                 sobral              2051     1229104      143536    12491032     80270316    11243145,00  72250085,00    Dec 07 13:10
at-spi-bus-laun      sobral              2162      305408        6560       22403         1139       20163,00     1026,00    Dec 07 13:10
at-spi2-registr      sobral              2249      162828        7480       36047        57434       32443,00    51691,00    Dec 07 13:10
bash                 sobral             16422       11240        5528  1395927332     8796608  1256334599,00  7916948,00    Dec 07 13:32
code                 sobral              2877     4725508      155712     7091780     10480306     6382602,00   9432276,00    Dec 07 13:10
code                 sobral              2881      186276       40492       81837           20       73654,00        18,00    Dec 07 13:10
code                 sobral              2882      186276       40848       81837            4       73654,00         4,00    Dec 07 13:10
code                 sobral              2912      435196       98352      250476      1165126      225434,00   1048614,00    Dec 07 13:10
code                 sobral              2925      258448       59456      541949      1010509      487755,00    909459,00    Dec 07 13:10
code                 sobral              2937    19480136      248112    53748755      5645392    48385809,00   5081013,00    Dec 07 13:10
code                 sobral              2997     4614044      361940  1054983172     26290252   951279361,00  23734330,00    Dec 07 13:10
code                 sobral              3012     4465340       67464      951078         8029      855971,00     7227,00    Dec 07 13:10
code                 sobral              3027     4464316       64936      742675         3282      668408,00     2954,00    Dec 07 13:10
code                 sobral              3104     4630776      131808    61993780     11907233    55796689,00  10716611,00    Dec 07 13:10
code                 sobral              4574     4423336       61916      332670          769      299403,00      693,00    Dec 07 13:11
dbus-daemon          sobral              1966        8940        6164      205566        18676      185010,00    16809,00    Dec 07 13:10
dbus-daemon          sobral              2167        7596        4504       59622        10424       53660,00     9382,00    Dec 07 13:10
dconf-service        sobral              2282      156464        5724       38482       332648       34634,00   299384,00    Dec 07 13:10
evolution-addre      sobral              2288      747856       30136      132231        43313      119008,00    38982,00    Dec 07 13:10
evolution-alarm      sobral              2387      785620       59948      386937        11233      348244,00    10110,00    Dec 07 13:10
evolution-calen      sobral              2273      913344       31088      159608        17586      143648,00    15828,00    Dec 07 13:10
evolution-sourc      sobral              2264      391184       26044      122917         4225      110626,00     3803,00    Dec 07 13:10
firefox              sobral              2639     3776800      502472   451744953    368961806   406580000,00 332075983,00    Dec 07 13:10
gdm-x-session        sobral              2049      164268        6964       38857          754       34972,00      679,00    Dec 07 13:10
gjs                  sobral              2303     2930780       27124       57592          689       51833,00      621,00    Dec 07 13:10
glib-pacrunner       sobral              2628      310136        7936      222794          557      200515,00      502,00    Dec 07 13:10
gnome-disks          sobral             77259      663488       50572      780437       960207      702394,00   864187,00    Dec 07 14:05
gnome-session-b      sobral              2073      188536       14332     5608534        23101     5047681,00    20791,00    Dec 07 13:10
gnome-session-b      sobral              2184      485648       17040      646338        34127      581705,00    30715,00    Dec 07 13:10
gnome-session-c      sobral              2177       90196        4360       24460           64       22014,00       58,00    Dec 07 13:10
gnome-shell          sobral              2198     4891028      306248    19196636     19797409    17277349,00  17822437,00    Dec 07 13:10
gnome-shell-cal      sobral              2255      581556       20704      127683         5690      114915,00     5121,00    Dec 07 13:10
gnome-terminal-      sobral             16363      818860       55680     1713509     11134052     1542159,00  10024499,00    Dec 07 13:32
goa-daemon           sobral              2008      546632       37296      127137         9210      114424,00     8289,00    Dec 07 13:10
goa-identity-se      sobral              2015      315264        9100      377773         7817      339996,00     7036,00    Dec 07 13:10
gsd-a11y-settin      sobral              2327      310184        6824       27202         8281       24482,00     7453,00    Dec 07 13:10
gsd-color            sobral              2328      585260       27160      318921        27608      287029,00    24848,00    Dec 07 13:10
gsd-datetime         sobral              2329      381928       25808     2076323        10164     1868691,00     9148,00    Dec 07 13:10
gsd-disk-utilit      sobral              2364      231792        5784       25143         2164       22629,00     1948,00    Dec 07 13:10
gsd-housekeepin      sobral              2331      312380        8064     1678282         8929     1510454,00     8037,00    Dec 07 13:10
gsd-keyboard         sobral              2338      342712       25684      292291        15508      263062,00    13958,00    Dec 07 13:10
gsd-media-keys       sobral              2340      898096       28436      326601        71151      293941,00    64036,00    Dec 07 13:10
gsd-power            sobral              2341      757396       28880      372977        22663      335680,00    20397,00    Dec 07 13:10
gsd-print-notif      sobral              2343      248720       11600       58350         8666       52515,00     7800,00    Dec 07 13:10
gsd-printer          sobral              2505      342464       15044       77255          656       69530,00      591,00    Dec 07 13:10
```

Figura 24 - ./procstat.sh 10

Os resultados esperados para esta execução são uma listagem de todos os processos que temos permissões, sendo isso que se verifica.

[2] Por questões de visualização e do espaço que as figuras preenchem, em algumas figuras não se encontram todos os processos guardados no array.

Relativamente a este teste (Fig.25), o esperado é que só sejam listados os processos que têm "c" no *comm*, e que sejam ordenados pelo RSS. Sendo que isso verifica-se.

```
sobral@sobral:~/Desktop/2ANO/SO/SO_Trabalho01$ ./procstat.sh -t -c "c.*" 10
COMM                 USER                 PID          MEM          RSS        READB       WRITEB        RATER        RATEW             DATE
code                 sobral              2997      4614044       361708   1499699786     44294763 1352442572,00  39974796,00 Dec 07 13:10
code                 sobral              2937     15253564       226768     63444328      5701722  57112089,00   5131770,00 Dec 07 13:10
code                 sobral              2877      4725508       156108      7095540     10888847   6385986,00   9799963,00 Dec 07 13:10
code                 sobral              3104      4630776       131544     84240298     18073757  75816269,00  16266382,00 Dec 07 13:10
code                 sobral              2912       434444       100692       251341      1166092    226207,00   1049483,00 Dec 07 13:10
code                 sobral              3012      4465340        68044       951078         8029    855971,00      7227,00 Dec 07 13:10
code                 sobral              4574      4431532        63560       332678          769    299411,00       693,00 Dec 07 13:11
code                 sobral              2925       258448        59544       715285      1320807    643757,00   1209751,00 Dec 07 13:10
code                 sobral              2882       186276        40848        81837            4     73654,00         4,00 Dec 07 13:10
code                 sobral              2881       186276        40492        81837           20     73654,00        18,00 Dec 07 13:10
RDD_Process          sobral             15494       186440        37880       138157         1345    124342,00      1211,00 Dec 07 13:23
evolution-calen      sobral              2273       913344        31092       164612        24546    148151,00     22092,00 Dec 07 13:10
gsd-color            sobral              2328       585260        27160       320025        29918    288023,00     26927,00 Dec 07 13:10
evolution-sourc      sobral              2264      1071132        26420       123889         5833    111501,00      5250,00 Dec 07 13:10
tracker-miner-f      sobral              1962       512396        25300     17912275        27795  16121048,00     25016,00 Dec 07 13:10
gsd-wacom            sobral              2356       268144        24712       431394        17460    388255,00     15714,00 Dec 07 13:10
gnome-shell-cal      sobral              2255       581684        20844       143231         8346    128908,00      7512,00 Dec 07 13:10
gvfs-afc-volume      sobral              2030       316996         8956        36436         9310     32793,00      8379,00 Dec 07 13:10
gsd-smartcard        sobral              2347       315684         8888        39340         9277     35406,00      8350,00 Dec 07 13:10
glib-pacrunner       sobral              2628       310136         7936       222874          717    200587,00       646,00 Dec 07 13:10
gsd-usb-protect      sobral              2352       385488         7668        30514         9899     27463,00      8910,00 Dec 07 13:10
ibus-memconf         sobral              2231       163200         7640        22986          576     20688,00       519,00 Dec 07 13:10
xdg-document-po      sobral              2479       457760         6848        57636        37930     51873,00     34137,00 Dec 07 13:10
gsd-screensaver      sobral              2345       235780         6056        33332        19972     29999,00     17975,00 Dec 07 13:10
dconf-service        sobral              2282       156464         5728        39314       417203     35383,00    375483,00 Dec 07 13:10
gnome-session-c      sobral              2177        90196         4360        24460           64     22014,00        58,00 Dec 07 13:10
procstat.sh          sobral            118835         9760         4052       155901         5063   7706673,00    175511,00 Dec 07 14:51
procstat.sh          sobral             94834         9632         3716      5543077       157815   4988770,00    142034,00 Dec 07 14:15
```

Figura 25 - ./procstat.sh -t -c "c." 10*

No seguinte teste (Fig.26), usamos o comando "-p", sendo que são esperados que só sejam listados o número de processos passados no terminal, ao correr o *script*.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -p 5 10
COMM                 USER                 PID          MEM          RSS        READB       WRITEB        RATER        RATEW             DATE
Xorg                 daniel              1641       945180        81888      4993235     29352365   4494336.50  26422361.50 Dec 07 21:42
at-spi-bus-laun      daniel              1746       305552         6684        22399         1099     20159.10       989.10 Dec 07 21:42
at-spi2-registr      daniel              1834       162828         7708        34191        42506     30771.90     38255.40 Dec 07 21:42
bash                 daniel              3325        11276         5384    965820417      5372831 869238375.30   4835547.90 Dec 07 21:44
cat                  daniel              2798         8228          520         4292            0      3862.80            0 Dec 07 21:43
```

Figura 26 - ./procstat.sh -p 5 10

No teste seguinte (Fig.27), passamos a opção "-s", e de seguida um argumento, sendo que o esperado são processo só com data superior em relação à data do processo.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -s "Dec 07 21:44" 10
COMM                 USER                 PID          MEM          RSS        READB       WRITEB        RATER        RATEW             DATE
bash                 daniel              3325        11276         5384   1091123069      6068343 982010762.10  5461508.70 Dec 07 21:44
chrome               daniel             23016      4776240       208860     18124559      4071559  16322361.10  3664410.10 Dec 07 21:50
chrome               daniel             33160       274540        58028       106100            6     95560.00        5.40 Dec 07 21:54
chrome               daniel             33294      4581236        54000        12993           12     13126.70       10.80 Dec 07 21:57
code                 daniel              3226      5040140       158848      9412606      8078797   8471347.40  7270919.30 Dec 07 21:44
code                 daniel              3229       186276        40848        81837           20     73653.30       18.00 Dec 07 21:44
code                 daniel              3230       186276        40872        81837            4     73653.30        3.60 Dec 07 21:44
code                 daniel              3252       485156       112348       260176      1433490    234162.40  1290145.00 Dec 07 21:44
code                 daniel              3272       257108        59140       715248        91197    643723.20    82077.30 Dec 07 21:44
code                 daniel              3284     19493640       250104     40398790      5067580  36378163.00  4561094.00 Dec 07 21:44
code                 daniel              3332      4551380       144196    256408657      9230958 233854236.30  8416529.20 Dec 07 21:44
code                 daniel              3347      4457144        66324       946942         3101    852247.80     2790.90 Dec 07 21:44
code                 daniel              3446      4456120        64556       730982         1204    657883.80     1083.60 Dec 07 21:44
code                 daniel              3472      4623280       115220      9747454      3638695   8772708.60  3274825.50 Dec 07 21:44
gnome-calendar       daniel             34931       854368        55460       377667         9274    339900.30     8346.60 Dec 07 21:59
gvfsd-dnssd          daniel             28071       315256         8700        34827         2024     31344.30     1821.60 Dec 07 21:51
gvfsd-network        daniel             23129       314632         8932        29901         2969     26910.90     2672.10 Dec 07 21:51
gvfsd-smb-brows      daniel             23137       489632        20552       127292         1718    114562.80     1546.20 Dec 07 21:51
new.sh               daniel              3612         9632         3600      5188878       147240   4669990.20   132516.00 Dec 07 21:45
new.sh               daniel              4852         9632         3452      4857420       138563   4371678.00   124706.70 Dec 07 21:46
new.sh               daniel              6006         9632          428            0            0         0.00        0.00 Dec 07 21:46
new.sh               daniel              6142         9632         3488      5364774       152230   4828296.60   137007.00 Dec 07 21:48
new.sh               daniel             12133         9632         3428      5482359       155555   4934123.10   139999.50 Dec 07 21:49
oosplash             daniel             33335       161848         4884       118641          516    106776.90      464.40 Dec 07 21:58
procstat.sh          daniel             53764         9764         4044      5798910       165871 118736352.00   597413.90 Dec 07 22:02
sleep                daniel              4846         8084          584         4292            0      3862.80            0 Dec 07 21:45
sleep                daniel              7417         8084          584         4292            0      3862.80            0 Dec 07 21:48
sleep                daniel             13436         8084          584         4292            0      3862.80            0 Dec 07 21:49
soffice.bin          daniel             33366      1331108       299336     17766301     15977347  15989670.90  14379612.30 Dec 07 21:58
```

Figura 27 - ./procstat.sh -s "Dec 07 21:44" 10

Este seguinte teste (Fig.28), é basicamente igual ao teste precedente, só que para a opção "-e", são esperados todos os testes que tenham uma data de começo inferior à data passada com a opção.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -e "Dec 07 21:44" 10
COMM                 USER                 PID         MEM         RSS       READB       WRITEB       RATER       RATEW            DATE
Xorg                 daniel              1641      965508       85136     5346604    33719299     4838309.60   31201623.10      Dec 07 21:42
at-spi-bus-laun      daniel              1746      305552        6684       22399        1099       20159.10       989.10      Dec 07 21:42
at-spi2-registr      daniel              1834      162828        7708       35359       46754       32095.10     43054.60      Dec 07 21:42
bash                 daniel              3325       11276        5384  1214096945     6711630  1092687250.50   6040467.00      Dec 07 21:44
cat                  daniel              2798        8228         520        4292           0        3862.80         0.00      Dec 07 21:43
cat                  daniel              2799        8228         524        5990        1698        5391.00      1528.20      Dec 07 21:43
chrome               daniel              2791     1088600      432952    40167004    32459367    36151461.60  29229293.30      Dec 07 21:43
chrome               daniel              2802      276160       63884      179908          21      161917.20        18.90      Dec 07 21:43
chrome               daniel              2803      276160       64100      179922          56      161929.80        50.40      Dec 07 21:43
chrome               daniel              2807      276160       16220      469618      215140      422656.20    193626.00      Dec 07 21:43
chrome               daniel              2826      588120      130904      344469     1272349      310965.10    1151572.10     Dec 07 21:43
chrome               daniel              2831      361556       96580    23741423    16201701    21368288.70  14581531.90      Dec 07 21:43
chrome               daniel              2849      318548       35276      511590       38504      460431.00     34653.60      Dec 07 21:43
chrome               daniel              2909     4654480      112360      839298       59726      765482.20     53763.40      Dec 07 21:43
chrome               daniel              3076    19605052      275760    74800640     7581659    67360802.00   6838601.10      Dec 07 21:43
chrome               daniel              3199     1107944       62160     7092258     5485403     6457648.20   4995926.70      Dec 07 21:43
code                 daniel              3226     5040140      158832    10261008     8123106     9235963.20   7312603.40      Dec 07 21:44
```

Figura 28 - ./procstat.sh -e "Dec 07 21:44" 10

Neste teste (Fig.29), o esperado é que só sejam listados os processos que começaram depois da data expressa na opção "-s" e antes da data expressa na opção "-e", também passamos a opção "-p" com o argumento 3, sendo que só queremos que sejam listados 3 processos, ainda passamos a opção "-w" que faz com que a lista apareça ordenada pela coluna do RATEW.

```
sobral@sobral:~/Desktop/2ANO/SO/SO_Trabalho01$ ./procstat.sh -w -s "Dec 07 13:30" -e "Dec 07 14:00" -p 3 10
COMM                 USER                 PID         MEM         RSS       READB       WRITEB       RATER       RATEW            DATE
gnome-terminal-      sobral             16363      819696       56188     2174865    19958618     1977326,00   18322513,00      Dec 07 13:32
bash                 sobral             16422       11408        5696  2312522310    14502555  2081270079,00  13052300,00      Dec 07 13:32
gvfsd-network        sobral             31091      388356        9352       33245        8105       29921,00      7295,00      Dec 07 13:37
```

Figura 29 - ./procstat.sh -w -s "Dec 07 13:30" -e "Dec 07 14:00" -p 3 10

Neste teste (Fig.30) é utilizada a opção "-u ", sendo que só aparecerão os processos em que o *user corresponda ao argumento passado na opção "-u".*

```
sobral@sobral:~/Desktop/2ANO/SO/SO_Trabalho01$ ./procstat.sh -u sobral 10
COMM                 USER                 PID         MEM         RSS       READB       WRITEB       RATER       RATEW            DATE
Privileged_Cont      sobral              2838     2471924      159720     2506615       21051     2255956,00     18948,00      Dec 07 13:10
RDD_Process          sobral             15494      186440       37880      138054        1242      124249,00      1118,00      Dec 07 13:23
WebExtensions        sobral              2767     2854576      306040     3141250       59039     2827150,00     53161,00      Dec 07 13:10
Web_Content          sobral              3176     2717748      327480     7463268      245070     6718579,00    222200,00      Dec 07 13:10
Web_Content          sobral              3401     2728464      374060     3901598       98205     3511504,00     88450,00      Dec 07 13:10
Web_Content          sobral              4394     3485164      740792     3779178      449039     3401659,00    404534,00      Dec 07 13:11
Web_Content          sobral             16118     2411468       92076     1063333        7388      957004,00      6654,00      Dec 07 13:29
Xorg                 sobral              2051     1230528      144956    15053264   100762344    13549090,00  90692766,00      Dec 07 13:10
at-spi-bus-laun      sobral              2162      305408        6560       22403        1139       20163,00      1026,00      Dec 07 13:10
at-spi2-registr      sobral              2249      162828        7480       36823       65346       33141,00     58812,00      Dec 07 13:10
bash                 sobral             16422       11408        5696  1589153682    10092117  1430238314,00   9082906,00      Dec 07 13:32
code                 sobral              2877     4725508      155972     7093280    10699718     6383952,00   9629747,00      Dec 07 13:10
code                 sobral              2881      186276       40492       81837          20       73654,00        18,00      Dec 07 13:10
code                 sobral              2882      186276       40848       81837           4       73654,00         4,00      Dec 07 13:10
code                 sobral              2912      434444      100692      250641     1166092      225577,00   1049483,00      Dec 07 13:10
code                 sobral              2925      258448       59472      577494     1089683      519745,00    980715,00      Dec 07 13:10
code                 sobral              2937    15253564      225556    56351722     5661441    50731567,00   5095517,00      Dec 07 13:10
code                 sobral              2997     4614044      361400   112662840    29247283  1016702906,00  26432172,00      Dec 07 13:10
code                 sobral              3012     4465340       67660      951078        8029      855971,00      7227,00      Dec 07 13:10
code                 sobral              3104     4630776      131604    66951659    13281116    60256494,00  11953005,00      Dec 07 13:10
code                 sobral              4574     4431532       63560      332678         769      299411,00       693,00      Dec 07 13:11
dbus-daemon          sobral              1966        8940        6164      214416       19776      192975,00     17799,00      Dec 07 13:10
dbus-daemon          sobral              2167        7596        4504       66662       13449       59996,00     12105,00      Dec 07 13:10
dconf-service        sobral              2282      156464        5728       38986      416651       35088,00    374986,00      Dec 07 13:10
evolution-addre      sobral              2288      747856       30136      132631       44001      119368,00     39601,00      Dec 07 13:10
evolution-alarm      sobral              2387      785620       59948      387153       11757      348438,00     10582,00      Dec 07 13:10
evolution-calen      sobral              2273      913344       31088      161242       19594      145118,00     17635,00      Dec 07 13:10
evolution-sourc      sobral              2264     1071132       26420      123473        5145      111126,00      4631,00      Dec 07 13:10
firefox              sobral              2639     4059220      890296   603878066   480125289  543492981,00 432115254,00      Dec 07 13:10
gdm-x-session        sobral              2049      164268        6964       38857         754       34972,00       679,00      Dec 07 13:10
gjs                  sobral              2303     2930780       27124       57592         689       51833,00       621,00      Dec 07 13:10
glib-pacrunner       sobral              2628      310136        7936      222794         557      200515,00       502,00      Dec 07 13:10
gnome-control-c      sobral            100352      194116       23044      343090        5125      308781,00      4613,00      Dec 07 14:15
gnome-session-b      sobral              2073      188536       14332     5608534       23101     5047681,00     20791,00      Dec 07 13:10
gnome-session-b      sobral              2184      485952       17300      651158       35495      586043,00     31946,00      Dec 07 13:10
```

Figura 30 - ./procstat.sh -u sobral 10

Neste teste (Fig.31), passamos várias opções "-m" e "-r", para que a informação seja apresenta por ordem crescente, "-s" e "-e" para data de início e de fim, "-u" para serem todos do utilizador passado como argumento, e "-p" para que só sejam listados o número passado como argumento nessa opção, processos.

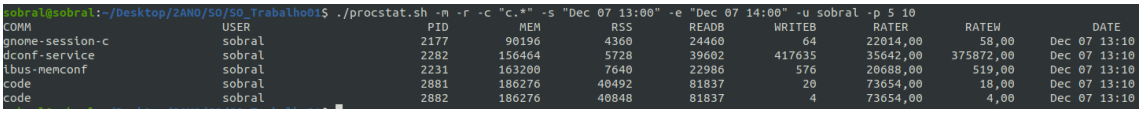


Figura 31 - ./procstat.sh -m -r -c "c." -s "Dec 07 13:00" -e "Dec 07 14:00" -u sobral -p 5 10*

5.2 – Resultados para opções/argumentos inválidos

Nesta parte do relatório, iremos exemplificar através das capturas feitas ao terminal, quando as opções/argumentos não são válidas.

O primeiro teste (Fig.32), passamos mal o argumento para os segundos, passamos uma *string*, em vez de um número, neste caso aparece a mensagem que o último argumento a ser passado terá de ser um número, e é chamada também a função *opcoes*, que mostra as maneiras de uso das opções.

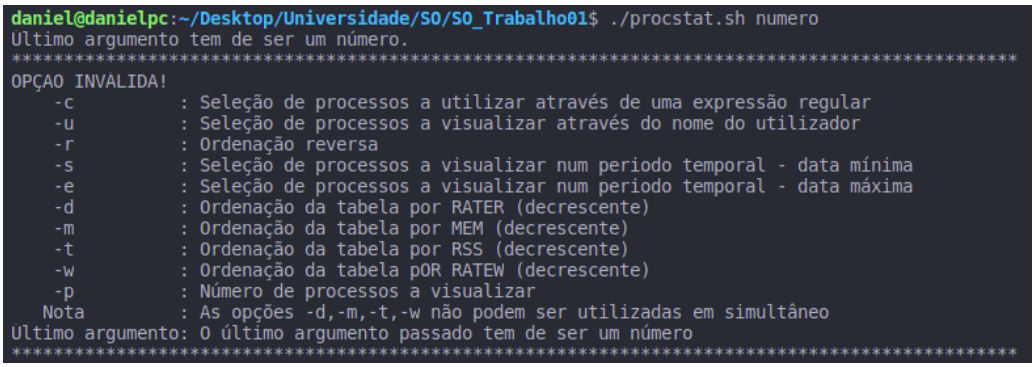


Figura 32 - Segundos inválidos

Neste teste (Fig.33), passamos mal a ordem dos argumentos da opção "-s", ao invés de **mês-dia hora:minutos**, passamos **dia-mês hora:minutos**, visto que a data não se encontra no formato desejado, aparece uma mensagem de erro, e a função *opcoes* é chamada.

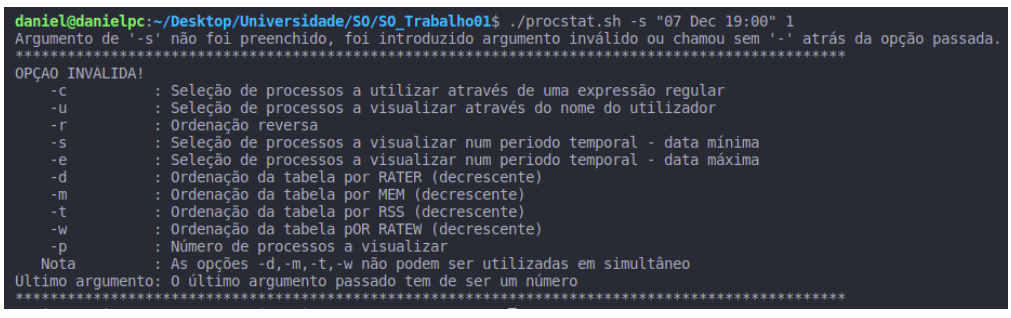


Figura 33 - Troca de argumentos na opção "-s"

No seguinte teste (Fig.34), executamos a opção "-c" de forma correta, mas o número de segundos introduzidos ao invés de um número, introduzimos uma string. O resultado foi o esperado, uma mensagem de erro, e a chamada da função *opcoes.*

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -c "cod.*" numero
Ultimo argumento tem de ser um número.
**********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
**********************************************************************************************
```

Figura 34 - opção "-c" bem, mas segundos mal

No próximo teste (Fig.35), são passados argumentos às opções "-s" e "-e", sendo que na opção "-e", a hora é passada num formato inválido, em vez de um ":", tem um "-". Como esperado é impressa uma mensagem de erro, e a função *opcoes.*

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -s "Dec 07 19:00" -e "07 Dec 19-10" 1
Argumento de '-e' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada.
**********************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
**********************************************************************************
```

Figura 35 - Argumento da opção "-e" mal passado

Neste teste (Fig.36), corremos o *script procstat.sh*, sem nenhuma opção ou argumento, sendo que neste caso não há informações suficientes para que a tabela seja gerada, nesse sentido aparece uma mensagem de erro, e é chamada a função *opcoes.*

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh
Tem de passar no mínimo um argumento (segundos).
**********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
**********************************************************************************************
```

Figura 36 - ./procstat.sh sem argumentos

No seguinte teste (Fig.37), é mostrado que quando é passado mais que um argumento de ordenação, a função *opcoes* é chamada. Isto acontece, pois não é possível ordenar mais que uma coluna cada vez que se corre o *script*.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -m -r -t 10
********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
********************************************************************************************
```

Figura 37 - Mais que uma opção de ordenação

Neste teste (Fig.38), testamos por o nome de um mês que não existe “Decem”, e nesse caso também aparece uma mensagem de erro, e a função *opcoes*, é chamada.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -s "Dec 07 19:00" -e "Decem 07 19:10" 1
Argumento de '-e' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada.
********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
********************************************************************************************
```

Figura 38 - Mês mal passado

No seguinte teste (Fig.39), passamos opções para ordenação “-m”, e “-r”, que são opções para listar os processos por ordem crescente, porém não foram passados o número de segundos, pois são um argumento obrigatório.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -m -r
********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
********************************************************************************************
```

Figura 39 - Ordenação sem número de segundos

No seguinte teste (Fig.40), usamos a opção "-c", no entanto não lhe passamos nenhum argumento a seguir, deste modo aparece uma mensagem de erro, e a função *opcoes* é chamada.

```
daniel@danielpc:~/Desktop/Universidade/SO/SO_Trabalho01$ ./procstat.sh -c -r 10
Argumento de '-c' não foi preenchido, foi introduzido argumento inválido ou chamou sem '-' atrás da opção passada.
********************************************************************************************
OPÇAO INVALIDA!
    -c          : Seleção de processos a utilizar através de uma expressão regular
    -u          : Seleção de processos a visualizar através do nome do utilizador
    -r          : Ordenação reversa
    -s          : Seleção de processos a visualizar num periodo temporal - data mínima
    -e          : Seleção de processos a visualizar num periodo temporal - data máxima
    -d          : Ordenação da tabela por RATER (decrescente)
    -m          : Ordenação da tabela por MEM (decrescente)
    -t          : Ordenação da tabela por RSS (decrescente)
    -w          : Ordenação da tabela pOR RATEW (decrescente)
    -p          : Número de processos a visualizar
   Nota         : As opções -d,-m,-t,-w não podem ser utilizadas em simultâneo
Ultimo argumento: O último argumento passado tem de ser um número
********************************************************************************************
```

Figura 40- Opção "-c" sem argumento

Conclusão

O *script* em questão, *procstat.sh*, permite visualizar as estatísticas dos processos que estão em execução na nossa máquina, e mais que isso, tal como pedido, permite filtrar a informação através de opções passadas no terminal no momento de execução do *script*.

Este trabalho, serviu para elucidar em muito os nossos conhecimentos, primeiramente aprendemos a trabalhar com arrays associativos, que são uma ferramenta poderosíssima, e que são bastante uteis, pois permitem guardar informação de acordo com uma *key*, alargamos também os nossos conhecimentos em relação ao *path* das diretorias/pastas, pois tivemos de ir buscar várias informações dentro de diretorias/pastas específicas dadas no guião, por último podemos dizer que aprendemos também a lidar com o acesso e permissões de certas e determinadas diretorias/pastas.

Ao longo da realização deste trabalho, foram aparecendo algumas dificuldades, sendo sinceros, no início estávamos ambos um pouco perdidos, porém depois de algum tempo a tentar perceber o que o trabalho realmente pedia, e como através de programação em *bash*, se conseguia responder às perguntas que o guião propunha, foi-nos fácil implementar o código.

Neste ponto da realização do trabalho/relatório podemos afirmar que conseguimos alcançar todos os objetivos que o guião propunha, sendo desta forma muito satisfatória a realização deste trabalho prático.

Bibliografia

- Para a realização deste trabalho prático, consultamos os slides disponibilizados pelo Professor na página do *e-learning* da unidade curricular Sistemas Operativos, bem como as aulas gravadas disponíveis no *e-learning*, sendo a mais vista a qual em que o Professor faz a introdução ao trabalho prático.

Sites visitados entre 27/12/2020 e 5/12/2020:

https://man.cx/bash

https://stackoverflow.com/

https://www.cyberciti.biz/faq/linux-unix-formatting-dates-for-display/

https://unix.stackexchange.com/questions/176198/determine-the-owner-of-the-session-of-a-process

https://www.linuxforce.com.br/comandos-linux/comandos-linux-comando-printf/