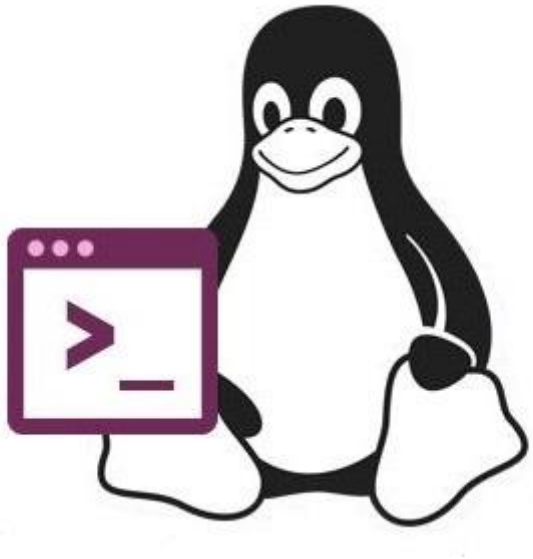
deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática



Trabalho 1

Taxas de Leitura/Escrita de processos em bash

Sistemas Operativos

Professor Nuno Lau e Professor Guilherme Campos

Trabalho realizado por:

João Nuno da Silva Luís (107403) | 50%

Diana Raquel Rodrigues Miranda (107457) | 50%

Índice

Índice de imagens

Introdução

Este trabalho foi realizado no âmbito da disciplina de Sistemas Operativos do 2º ano da Licenciatura em Engenharia Informática.

Foi-nos proposto desenvolver um *script* em bash que obtivesse estatísticas de leitura e de escrita de processos, podendo assim ver o número total de bytes de I/O (Input/Output) que um processo leu/escreveu e posteriormente a taxa de leitura/escrita, correspondente ao número de segundos desejado, valor passado como parâmetro aquando da execução do programa.

Para além disto, foram propostos alguns filtros, que complementam a visualização de informação destes mesmos processos:

- -c: Filtro que permite, através de uma expressão regular, filtrar a seleção de processos pelo seu nome
- -u: Filtro que permite visualizar a seleção de processos pelo nome de utilizador
- -s: Ver os processos que iniciaram a sua execução após a data inserida como parâmetro neste filtro.
- -e: Ver os processos que iniciaram a sua execução antes da data inserida como parâmetro neste filtro.
- -m: Através de um PID mínimo passado como parâmetro, filtrar os processos com PID maior.
- -M: Através de um PID máximo passado como parâmetro, mostrar apenas os processos com PID menor.
- -w: Organizar a tabela em função da coluna RATEW, correspondente aos valores de escrita.
- -r: Mostrar a tabela em ordem contrária (crescente) à ordem *default.*

Metodologia

O nosso código está organizado da seguinte forma:

Começámos por fazer a função help() que tem a função de auxiliar o utilizador no uso das opções disponíveis para a pesquisa dos processos. De seguida declarámos as variáveis necessárias e fizemos a sua inicialização. Após termos as variáveis todas declaradas, procedemos à implementação do comando getops para fazer o tratamento das opções de entrada. E depois avançámos para a leitura dos valores e o tratamento de toda a informação.

Por fim, procedemos ao print da tabela com a informação organizada.

- **Declaração de Variáveis**

```
#Declarar as variáveis
declare -A final_info=()

arrPID=(); #array vazio
arrCOMM=(); #array vazio
arrUSER=() #array vazio
arrREAD1=() #array vazio
arrWRITE1=() #array vazio

#Buscar as informações necessárias
PID=$(ps -e -o pid | grep -v PID); #Vai buscar os valores dos PIDs dos processos em execução
COMM=$(ps -e -o comm | grep -v COMMAND); #Vai buscar o COMM dos processos em execução
USER=$(ps -e -o user | grep -v USER); #Vai buscar os users dos processos em execução
LSTART=$(ps -e -o lstart | grep -v STARTED); #Vai buscar a data de inicio dos processos em execução

#Cria o array com os PIDs
while read line
do
   [[ "$line" != '' ]] && arrPID+=("$line")
done <<< "$PID"
#Cria o array com os COMM
while read line
do
   [[ "$line" != '' ]] && arrCOMM+=("$line")
done <<< "$COMM"
#Cria o array com os Users
while read line
do
   [[ "$line" != '' ]] && arrUSER+=("$line")
done <<< "$USER"
#Cria o array com as datas de inicio
while read line
do
   [[ "$line" != '' ]] && arrLSTART+=("$line")
done <<< "$LSTART"

#Inicio das variaveis
nprocessos="${#arrPID[@]}"; # número de processos existentes (é a length do array)
colOrdena=6; #coluna que vai ser ordenada
procName=(.*); #nome do processo
userName=(.*); #nome do utilizador
minPid=0; #PID minimo
minPidFinal=(.*); #todos os PIDs superiores ao minPid
maxPid=0; #PID maximo
maxPidFinal=(.*); #todos os PIDs inferiores ao maxPid
sortmethod=(sort -k $colOrdena -n -r) # inicaliza o sort para ordenar por ordem decresecente de RATER.
minDate=0; #Data mínima
maxDate=0; #Data máxima
minDateFinal=(.*); #todas os datas superiores ao minDate
maxDateFinal=(.*); #todas os datas inferiores ao maxDate

if ! [[ ${@: -1} =~ ^[0-9]+$ ]]; then #Verifica se o último argumento é um número
   echo "ERRO: O último argumento tem de ser um número!"
   help
```

Figura 1 - Declaração de variáveis

Algumas variáveis foram inicializadas com valores *default,* facilitando a utilização dos argumentos de entrada no programa. Estas variáveis vão ser depois atualizadas, conforme as opções selecionadas pelo utilizador e utilizadas para devolver os resultados desejados.

Para o correto tratamento dos dados, decidimos usa um *array* associativo (**final_info**) já que estes permitem associar diferentes tipos de valores a uma chave, onde neste projeto o PID de cada processo é a chave.

Para as variáveis relacionadas com os filtros, definimos a variável **nprocessos** que é iniciada com o comprimento do *array* **arrPID,** ou seja, com o número de processos existentes, esta variável vai ser usada mais tarde para definir quantos processos vão ser impressos na tabela.

Inicializámos também a variável **sortmethod**, que usa o comando sort em função da variável **$colOrdena**, que corresponde ao número da coluna pela qual queremos ver os valores da tabela ordenados.

Já as restantes variáveis foram inicializadas ou com o valor 0, ou com o valor (.*), que significa que se não for passado nenhum valor, a variável vai aceitar qualquer caracter em qualquer quantidade.

- **Tratamento de argumentos**

Para o correto funcionamento dos diferentes filtros, usámos o comando *getopts,* que juntamente com as condições de validação e aplicação por nós acrescentadas, vai proceder ao tratamento dos dados e realizar as tarefas solicitadas pelos argumentos passados.

```
while getopts "c:u:m:M:s:e:rwp:" opt; do
   case $opt in
   c) procName=$OPTARG
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      elif [[ $procName =~ ^([0-9]+)$ ]]; then
         echo "ERRO: A opção -c requere um argumento ou um argumento diferente de um número"
         help
      fi
      ;;
   u) if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      elif [[ $OPTARG =~ ^[0-9]*$ ]]; then
         echo "A opção -u requere um argumento ou um que não seja um número"
      fi
      userName="^$OPTARG$";;
   m) minPid=$OPTARG
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      fi
      minPidFinal=(^[0-9]*$)
      if  [[ ! $minPid =~ ^([0-9]+)$ ]] || [[ $minPid -eq ${@: -1} ]]; then # se o minPidFinal não for um número inteiro, ou se for o valor passado como sleep, então dá erro e vai para a ajuda
         echo "ERRO: o número mínimo do ID do processo a visualizar tem de ser um inteiro positivo, e este não pode ser o valor passado como tempo de sleep."
         help
    fi;;
   M) maxPid=$OPTARG
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      fi
      maxPidFinal=(^[0-9]*$)
      if [[ ! $maxPid =~ ^([0-9]+)$ ]] || [[ $maxPid -eq ${@: -1} ]]; then # se o maxPidFinal não for um número inteiro, então dá erro e vai para a ajuda
         echo "ERRO: o número máximo do ID do processo a visualizar tem de ser um inteiro positivo, e este não pode ser o valor passado como tempo de sleep."
         help
    fi;;
   s) minDate=$OPTARG
      minDateFinal=(^[A-Z]*$)
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      elif [[ "${#OPTARG}" -ne 12 ]]; then #para ser uma data válida, tem de ter 12 caracteres a contar com os espaços
         echo "ERRO: a data mínima tem de ter 12 carateres (os espaços também contam)"
         help
      fi;;
   e) maxDate=$OPTARG
      maxDateFinal=(^[A-Z]*$)
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      elif [[ "${#OPTARG}" -ne 12 ]]; then #para ser uma data válida, tem de ter 12 caracteres a contar com os espaços
         echo "ERRO: o número máximo do ID do processo a visualizar tem de ser um inteiro positivo, e este não pode ser o valor passado como tempo de sleep."
         help
      fi;;
      #acrescentar REGEX para validar a data
   r) sortmethod=(sort -k $colOrdena -n);;
   w) colOrdena=7;
      sortmethod=(sort -k $colOrdena -n -r);;
   p) nprocessos=$OPTARG
      if [[ ${OPTARG:0:1} == "-" ]]; then # todos os comandos têm de começar por -
         echo "ERRO: está a passar como argumento outro comando!"
         help
      elif [[ ! $maxPid =~ ^([0-9]+)$ ]] || [[ $maxPid -eq ${@: -1} ]]; then # se o nprocessos não for um número inteiro, ou for o valor passado no sleep então dá erro e vai para a ajuda
         echo "ERRO: o número de processos a visualizar tem de ser um inteiro positivo."
         help
    fi
      ;;
   ?) help;;
   esac
done
```

Figura 2 - Código da implementação do getops

Para todas as opções que solicitam um argumento (exemplo: -p 5), definimos uma condição que impossibilitasse a passagem de outra opção como argumento, sendo impresso no terminal uma mensagem caso ocorra esse erro.

No caso da opção **-c e -u**, o valor passado na variável $OPTARG vai ser comparado com uma expressão regular que por sua vez mostra um erro se não for passado nenhum valor no $OPTARG ou se for passado um número.

Já no caso das opções -**m**, –**M** e -**p**, é gerado um erro se o argumento passado não for um inteiro positivo ou se esse valor for o último parâmetro, correspondente ao número de segundos a utilizar para a seleção de processos

No que toca às opções **-s** e –**e,** o valor passado entre aspas tem de ter exatamente 12 carateres (os espaços também contam), e tem de cumprir o comando *date*, ou seja, se for passado, por exemplo o mês “Nev” o programa irá devolver o erro de data inválida.

Por fim, nas opções -**r** e –**w**, em que não são necessários parâmetros extra, é feita a correta ordenação dos processos, com o recurso ao comando sort, havendo na segunda opção uma organização em função dos valores do RATEW, sendo por isso feito o ordenamento pela variável ***colOrdena***, que neste caso vai tomar o valor 7, referente à sétima coluna da tabela.

Para além destas opções, existe também a função ***help()***, que é chamada quando ocorre algum erro, ou quando é utilizada uma opção que não está listada no comando ***getopts***.

```
#Opções disponíveis
help() {
    echo "-------------------------------------------------------------------------------------------------"
    echo "| OPÇÕES DISPONÍVEIS!                                                                           |"
    echo "|                                                                                               |"
    echo "| Opções de visualização:                                                                       |"
    echo "|   -c    : Selecionar os processos a visualizar através de uma expressão regular               |"
    echo "|   -s    : Definir data mínima de criação dos processos a visualizar                           |"
    echo "|   -e    : Definir data máxima de criação dos processos a visualizar                           |"
    echo "|   -u    : Visualizar os processos de um determinado utilizador                                |"
    echo "|   -m    : Definir o PID mínimo dos processos a visualizar                                     |"
    echo "|   -M    : Definir o PID máximo dos processos a visualizar                                     |"
    echo "|   -p    : Definir o número de processos a visualizar                                          |"
    echo "|                                                                                               |"
    echo "| Opções de ordenação:                                                                          |"
    echo "|   -r    : Ordem reversa (crescente)                                                           |"
    echo "|   -w    : Ordenar pelo RATEW (descrescente)                                                   |"
    echo "|    A ordenação default é pelo RATER de forma decrescente.                                     |"
    echo "|                                                                                               |"
    echo "| O último argumento tem de corresponder sempre ao número de segundos que pretende analisar.    |"
    echo "-------------------------------------------------------------------------------------------------"
    exit 1
}
```

Figura 3 - Função help

- **Tratamento da pesquisa por data e regex**

No tratamento da pesquisa por datas (opções -s e –e) começámos por transformar, com recurso ao comando awk, a data de cada processo (variável **DATE_Segundos**), a data passada no argumento da opção **-s** (variável **inicio**) e a data passada em **–e** (variável **fim**) em segundos.

Quando é usada a opção **-s** é feita uma comparação que verifica se o número de segundos da data do processo é maior do que o número de segundos da data mínima que queremos. Se for, a data do processo é então concatenada a uma variável que guarda todas as datas que satisfazem este caso separadas por um "|". Esta variável serve para mais tarde fazer a procura de todas as datas válidas nesta condição através do comando **grep**.

Quando é utilizada a opção **-e,** é realizado o mesmo método descrito acima, com uma ligeira diferença que ocorre na comparação. Neste caso é verificado se o número de segundos da data do processo é menor do que o número de segundos da data máxima que queremos.

Para o funcionamento correto deste processo foi necessário exportar no início do código a linguagem pt_BR.utf8 (export LANG=pt_BR.utf8), pois o nosso computador tem o time local em português e o lang em inglês, o que estava a gerar conflito com o uso do comando **date** na transformação da data em segundos devido a diferenças como, por exemplo "Dec" em inglês e "Dez" em português.

```
DATE_Segundos=$(date -d "$DATE" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}') # data do processo em segundos

inicio=$(date -d "$minDate" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}')

if [[ $DATE_Segundos -ge $inicio ]]; then
   minDateFinal=$(echo "$minDateFinal"'|'"$DATE") #Concatena uma string com os PID para depois fazer o grep
fi

fim=$(date -d "$maxDate" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}')

if [[ $DATE_Segundos -le $fim ]]; then
   maxDateFinal=$(echo "$maxDateFinal"'|'"$DATE") #Concatena uma string com as datas para depois fazer o grep
fi
```

Figura 4 - Tratamento das datas para aplicação do filtro

No tratamento das expressões regex utilizámos o comando **awk** para identificar o padrão passado pelo utilizador na coluna correspondente. Isto é, se pretendesse fazer uma pesquisa pelo nome do processo o comando **awk** iria só procurar o padrão passado na coluna 1, e se a pesquisa fosse pelo utilizador, iria só procurar na coluna 2. Neste segundo caso, como queremos uma procura exata, acrescentámos ^ no início do argumento para indicar que era o início da *string* e que não pode ter nada antes desse padrão, e no fim acrescentámos $ para indicar que é o fim da *string*, ou seja, nada pode estar depois desse padrão (`userName="^$OPTARG$"`).

```
awk -v pat=$userName '$2 ~ pat' | awk -v pat=$procName '$1 ~ pat'
```

Figura 5 - Uso do regex para filtro através do user e do nome do processo

- **Extração e tratamento da informação**

Para a extração da informação começámos por correr o comando **ps -e** que lista toda a informação sobre os processos em funcionamento e guardámos essa informação nas respetivas variáveis.

```
#Buscar as informações necessárias
PID=$(ps -e -o pid | grep -v PID); #Vai buscar os valores dos PIDs dos processos em execução
COMM=$(ps -e -o comm | grep -v COMMAND); #Vai buscar o COMM dos processos em execução
USER=$(ps -e -o user | grep -v USER); #Vai buscar os users dos processos em execução
LSTART=$(ps -e -o lstart | grep -v STARTED); #Vai buscar a data de inicio dos processos em execução
```

Figura 6 - Código para a extração da informação necessária

Em seguida, executámos para cada PID guardado no array arrPID o comando "**cat /proc/PID/io**" para termos acesso ao número de caracteres escritos e lidos por esse processo. Depois, executámos o comando **sleep** com o número de segundos passado no último argumento e após isso voltámos a ler o número de caracteres escritos e lidos por cada processo.

Com esta informação, pudemos então fazer a diferença entre os valores lidos inicialmente e os valores lidos após x segundos. Seguidamente, procedemos ao cálculo da taxa de leitura (**RateW**) e de escrita (**RateR**) de cada processo.

```
for (( i=0; i<${#arrPID[@]}; i++ ))
    do
      #só dá print se o ficheiro existir
       if [ -r /proc/${arrPID[$i]}/io  ]; then
          #Vai para cada PID buscar os valores do rchar e do wchar
          arrREAD1[$i]=$(cat /proc/${arrPID[$i]}/io | grep rchar | awk '{print $2}');
          arrWRITE1[$i]=$(cat /proc/${arrPID[$i]}/io | grep wchar | awk '{print $2}');
       fi
done

sleep ${@: -1} # argumento do tempo. É sempre o ultimo a ser passado independentemente do nº de argumentos
```

Figura 9 - Leitura dos valores iniciais de rchar e wchar

```
for (( i=0; i <= ${#arrPID[@]}; i++ ))
    do
      #Verifica se tem permissões de leitura, se não tiver ignora
      if [ -r /proc/${arrPID[$i]}/io ]; then

        if [[ ${arrPID[$i]} -ge $minPid ]]; then
           minPidFinal=$(echo $minPidFinal'|'${arrPID[$i]}) #Concatena uma string com os PID para depois fazer o grep
        fi
        if [[ ${arrPID[$i]} -le $maxPid ]]; then
           maxPidFinal=$(echo $maxPidFinal'|'${arrPID[$i]})
        fi

        READB2=$(cat /proc/${arrPID[$i]}/io | grep rchar | awk '{print $2}');
        WRITEB2=$(cat /proc/${arrPID[$i]}/io | grep wchar | awk '{print $2}');
        LSTART=${arrLSTART[$i]};

        DATE=$(date -d "$LSTART" +"%b %d %H:%M" | awk '{$1=toupper(substr($1,0,1))substr($1,2)}1'); #O awk é para colocar a primeira letra do Mês maiúscula
        DATE_Segundos=$(date -d "$DATE" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}') # data do processo em segundos

        inicio=$(date -d "$minDate" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}')

        if [[ $DATE_Segundos -ge $inicio ]]; then
           minDateFinal=$(echo "$minDateFinal"'|'"$DATE") #Concatena uma string com os PID para depois fazer o grep
        fi

        fim=$(date -d "$maxDate" +"%b %d %H:%M"+%s | awk -F '[+]' '{print $2}')

        if [[ $DATE_Segundos -le $fim ]]; then
           maxDateFinal=$(echo "$maxDateFinal"'|'"$DATE") #Concatena uma string com as datas para depois fazer o grep
        fi

        READB=$(echo "($READB2 - ${arrREAD1[$i]})" | bc);
        WRITEB=$(echo "($WRITEB2 - ${arrWRITE1[$i]})" | bc);
        RATER=$(echo "scale=2; $READB/${@: -1}" | bc);
        RATEW=$(echo "scale=2; $WRITEB/${@: -1}" | bc);
```

Figura 8 - Segunda leitura dos valores rchar e wchar e cálculo do ReadB e do WriteB. E cálculo das taxas de leitura/Escrita (RateR e RateW)

Toda a informação necessária foi então guardada num *array* associativo (final_info) que tem como chave o PID de cada processo e como valores todas a informações necessárias a ele associadas, já formatadas.

```
        final_info[${arrPID[$i]}]=$(printf "%-20s %-12s %-12s %-12s %-12s %-12s %-12s %-12s" "${arrCOMM[$i]}" "${arrUSER[$i]}" "${arrPID[$i]}" "$READB" "$WRITEB" "$RATER" "$RATEW" "$DATE");
      fi
done
```

Figura 7 - Armazenamento de toda a informação formatada num array

Para impressão de toda a informação numa tabela organizada, executámos o comando **printf** para imprimir todos os valores guardados no *array* **final_info** e utilizamos os *pipes* para executar os filtros que fossem necessários utilizando os comandos **awk**, **grep** e **head**.

- O comando **awk –v pat=$variavel '$x ~ pat'** procura na coluna x o padrão indicado na variável;

- O comando **grep –E $variavel** interpreta o padrão indicado na variável como uma expressão regular estendida;
- O comando **head –n $x** filtra a informação para só aparecerem as primeiras x linhas.

```
#Prints
printf "%-20s %-12s %-12s %-12s %-12s %-12s %-12s %-12s \n" "COMM" "USER" "PID" "READB" "WRITEB" "RATER" "RATEW" "DATE";
printf "%s\n" "${final_info[@]}" | "${sortmethod[@]}" | awk -v pat=$userName '$2 ~ pat' | awk -v pat=$procName '$1 ~ pat' | grep -E $minPidFinal
    | grep -E $maxPidFinal | grep -E "$minDateFinal" | grep -E "$maxDateFinal" | head -n $nprocessos
```

Figura 10 - Print da tabela final

Testes

Mostramos agora algumas combinações possíveis, que podem ser feitas pelo utilizador na utilização deste programa, e que por nós foram também utilizadas, para verificar se obtínhamos os resultados desejados.

Assim sendo, começamos pela execução mais simples, em que apenas é passado como argumento o número de segundos.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh 2
COMM            USER        PID         READB       WRITEB      RATER       RATEW       DATE
rwstat.sh       joao        17036       17570685    206343      8785342.50  103171.50   Dec 01 23:16
code            joao        7696        349155      30          174577.50   15.00       Dec 01 22:31
code            joao        8030        199145      4570        99572.50    2285.00     Dec 01 22:31
brave           joao        6048        24355       9471        12177.50    4735.50     Dec 01 22:12
brave           joao        6611        4602        0           2301.00     0           Dec 01 22:12
```

Figura 11 - 1º teste. Execução do programa sem nenhum filtro

Obtemos uma tabela com os valores para cada processo, estando esta organizada pelos valores do RATER.

De seguida, usamos a opção **-c,** passando a expressão regular “d.*”, esperando assim visualizar todos os processos que contenham a letra d no seu nome. Já no segundo caso, testamos a opção **-u**, que devolve os processos cujo utilizador seja igual à expressão passada.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -c "d.*" 2
COMM            USER        PID         READB       WRITEB      RATER       RATEW       DATE
code            joao        8031        3803        426         1901.50     213.00      Dec 01 22:31
code            joao        7779        254         81          127.00      40.50       Dec 01 22:31
gsd-sharing     joao        2941        24          40          12.00       20.00       Dec 01 22:10
code            joao        8030        8           8           4.00        4.00        Dec 01 22:31
xdg-permission-  joao       2440        0           0           0           0           Dec 01 22:10
```

Figura 12 - 2º teste. Filtro pelo nome do processo

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -u joao 2
COMM            USER        PID         READB       WRITEB      RATER       RATEW       DATE
rwstat.sh       joao        33685       17570833    206416      8785416.50  103208.00   Dec 01 23:31
WD-TabNine      joao        8194        3171371     0           1585685.50  0           Dec 01 22:31
brave           joao        6611        14290       7           7145.00     3.50        Dec 01 22:12
code            joao        8031        4137        478         2068.50     239.00      Dec 01 22:31
brave           joao        10990       1535        0           767.50      0           Dec 01 23:06
```

Figura 13 - 3º teste. Filtro pelo nome de utilizador

```
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$ ./rwstat.sh -u di 2
COMM                 USER         PID          READB        WRITEB       RATER        RATEW        DATE
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$
```

Figura 14 - 3º teste. Filtro pelo nome de utilizador

Esta última imagem não mostra nenhum processo pois não existe nenhum utilizador com o nome “di”.

Agora, testamos as opções **-s** e –**e** em simultâneo, e em cada uma, passamos como argumento uma data mínima e máxima, respetivamente. Estas opção funcionam também em separado.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -s "Dec 01
 22:40" -e "Dec 01 23:10" 2
COMM                 USER         PID          READB        WRITEB       RATER        RATEW        DATE
gvfsd-network        joao         10392        0            0            0            0            Dec 01 23:01
gvfsd-dnssd          joao         10406        0            0            0            0            Dec 01 23:01
brave                joao         9722         0            0            0            0            Dec 01 22:57
brave                joao         10990        0            0            0            0            Dec 01 23:06
```

Figura 15 - 4º teste. Filtro por data

Já quanto às opções **-m** e –**M,** é passado um valor mínimo e máximo de PID. Vamos utilizar também a opção **-p**, de forma a limitar o número de processos a ver. Estas opções funcionam também em separado.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -m 5000 -M
 12000 -p 3 2
COMM                 USER         PID          READB        WRITEB       RATER        RATEW        DATE
brave                joao         6611         10239        4            5119.50      2.00         Dec 01 22:12
code                 joao         8031         4137         478          2068.50      239.00       Dec 01 22:31
brave                joao         6323         580          578          290.00       289.00       Dec 01 22:12
```

Figura 16 - 5º Teste. Filtro por número máximo e mínimo de PID e filtro no número de processos a visualizar

De notar que, apesar de todos estas opções serem usadas, os processos mantêm-se ordenados pelos valores de RATER.

Assim, vamos agora testar as opções **-w** e –**r,** em simultâneo com outras opções, para verificarmos resultados.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -w -c "d.*
" 2
COMM                 USER         PID          READB        WRITEB       RATER        RATEW        DATE
dbus-daemon          joao         2398         804          1398         402.00       699.00       Dec 01 22:10
code                 joao         8031         350          68           175.00       34.00        Dec 01 22:31
code                 joao         8072         0            31           0            15.50        Dec 01 22:31
code                 joao         8030         8            8            4.00         4.00         Dec 01 22:31
code                 joao         7779         21           8            10.50        4.00         Dec 01 22:31
```

Figura 17 - 6º teste. Ordenação da tabela pelo RATEW

```
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$ ./rwstat.sh -r -c "br.*" 2
COMM                 USER         PID          READB        WRITEB       RATER        RATEW        DATE
brave                diana        5092         0            0            0            0            Dec 02 14:11
brave                diana        5093         0            0            0            0            Dec 02 14:11
brave                diana        5095         0            0            0            0            Dec 02 14:11
brave                diana        5137         0            0            0            0            Dec 02 14:11
brave                diana        5154         0            0            0            0            Dec 02 14:11
brave                diana        5420         0            0            0            0            Dec 02 14:11
brave                diana        5692         0            0            0            0            Dec 02 14:11
brave                diana        5783         0            0            0            0            Dec 02 14:11
brave-browser        diana        4936         0            0            0            0            Dec 02 14:11
brave                diana        5068         203          251          101.50       125.50       Dec 02 14:11
brave                diana        5127         413          3            206.50       1.50         Dec 02 14:11
brave                diana        5265         485          484          242.50       242.00       Dec 02 14:11
brave                diana        5170         1517         0            758.50       0            Dec 02 14:11
brave                diana        39125        1519         0            759.50       0            Dec 02 14:20
brave                diana        5300         4631         6            2315.50      3.00         Dec 02 14:11
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$
```

Figura 18 - 7º teste. Ordenação da tabela na ordem reversa (crescente pelo RATER)

Por último, vamos testar combinar bastantes opções ao mesmo tempo.

```
(base) joao@joao-IdeaPad-5-15ARE05:~/Documents/Ano2/Semestre1/SO-Sistemas_Operativos/SO_Trabalho1$ ./rwstat.sh -w -r -c "
d.*" -u joao -m 5000 -M 10000 -s "Dec 01 22:15" -e "Dec 01 23:10" 2
COMM                USER         PID          READB        WRITEB       RATER        RATEW        DATE
TabNine-deep-lo     joao         8338         0            0            0            0            Dec 01 22:31
chrome_crashpad     joao         7711         0            0            0            0            Dec 01 22:31
code                joao         7696         0            0            0            0            Dec 01 22:31
code                joao         7698         0            0            0            0            Dec 01 22:31
code                joao         7699         0            0            0            0            Dec 01 22:31
```

Figura 19 - 8º teste. Execução do programa com vários filtros aplicados

Posto isto, depois de todas as opções de filtragem testadas confirmamos que todas funcionam como esperado, tendo tido sucesso em todos os testes.

Erros

Nesta secção mostramos algumas mensagens de erro que poderão aparecer ao utilizador se não executar o programa ou as opções de filtro corretamente.

```
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$ ./rwstat.sh
ERRO: O último argumento tem de ser um número!
-----------------------------------------------------------------------------------------------
| OPÇÕES DISPONÍVEIS!                                                                         |
|                                                                                             |
| Opções de visualização:                                                                     |
|   -c    : Selecionar os processos a visualizar através de uma expressão regular             |
|   -s    : Definir data mínima de criação dos processos a visualizar                         |
|   -e    : Definir data máxima de criação dos processos a visualizar                         |
|   -u    : Visualizar os processos de um determinado utilizador                              |
|   -m    : Definir o PID mínimo dos processos a visualizar                                   |
|   -M    : Definir o PID máximo dos processos a visualizar                                   |
|   -p    : Definir o número de processos a visualizar                                        |
|                                                                                             |
| Opções de ordenação:                                                                        |
|   -r    : Ordem reversa (crescente)                                                         |
|   -w    : Ordenar pelo RATEW (descrescente)                                                 |
|    A ordenação default é pelo RATER de forma decrescente.                                   |
|                                                                                             |
| O último argumento tem de corresponder sempre ao número de segundos que pretende analisar.  |
-----------------------------------------------------------------------------------------------
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$
```

Figura 21 - Tentativa de execução do programa sem argumentos

```
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$ ./rwstat.sh -p -u diana 2
ERRO: está a passar como argumento outro comando!
-----------------------------------------------------------------------------------------------
| OPÇÕES DISPONÍVEIS!                                                                         |
|                                                                                             |
| Opções de visualização:                                                                     |
|   -c    : Selecionar os processos a visualizar através de uma expressão regular             |
|   -s    : Definir data mínima de criação dos processos a visualizar                         |
|   -e    : Definir data máxima de criação dos processos a visualizar                         |
|   -u    : Visualizar os processos de um determinado utilizador                              |
|   -m    : Definir o PID mínimo dos processos a visualizar                                   |
|   -M    : Definir o PID máximo dos processos a visualizar                                   |
|   -p    : Definir o número de processos a visualizar                                        |
|                                                                                             |
| Opções de ordenação:                                                                        |
|   -r    : Ordem reversa (crescente)                                                         |
|   -w    : Ordenar pelo RATEW (descrescente)                                                 |
|    A ordenação default é pelo RATER de forma decrescente.                                   |
|                                                                                             |
| O último argumento tem de corresponder sempre ao número de segundos que pretende analisar.  |
-----------------------------------------------------------------------------------------------
diana@diana-Legion-5-Pro-16ACH6:~/LEI/SO/Projeto$
```

Figura 20 - Tentativa de execução sem passar argumento numa opção que assim o exigia

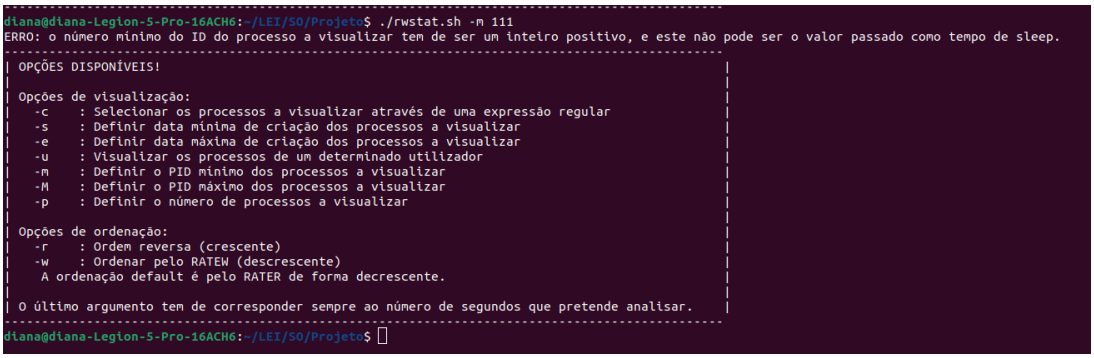


Figura 24 - Tentativa de execução sem argumento de tempo

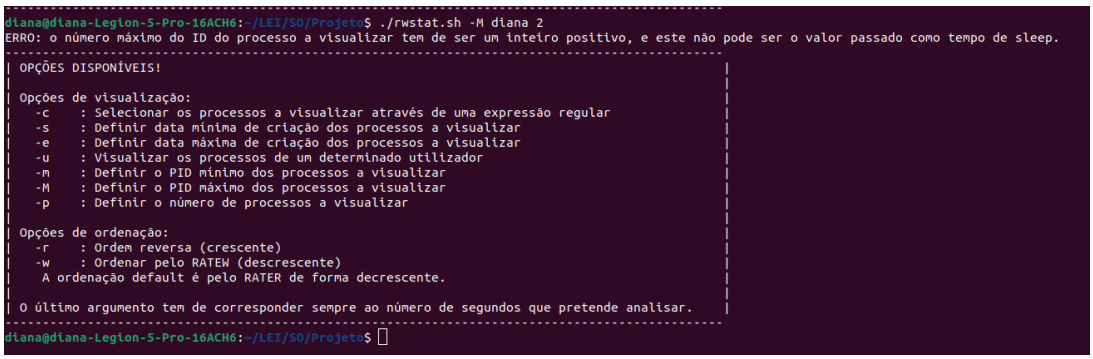


Figura 23 - Tentativa de execução com passagem de uma string numa opção que exigia números como argumento

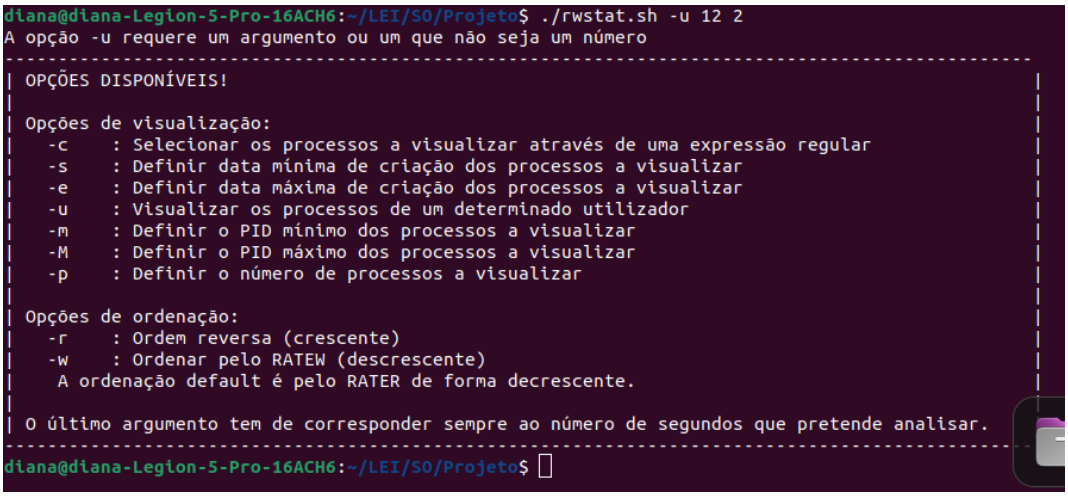


Figura 22 - Tentativa de execução com passagem de um número numa opção que exigia uma string como argumento

Conclusão

Com este trabalho, desenvolvemos bastante o nosso conhecimento sobre *Bash,* com o uso de novos comandos (por exemplo o getopts()), e também de algumas estruturas de dados, como os *arrays* associativos.

Para o uso correto desses mesmos comandos, surgiram algumas dúvidas, que foram maioritariamente respondidas com recurso a fóruns na internet ou com recurso ao comando "**man**" do terminal.

Para além disso, aprofundámos também o nosso conhecimento sobre Git e GitHub, visto que utilizámos ambos para o controlo de versões e partilha do trabalho entre os dois elementos do grupo.

Assim, concluímos que os objetivos deste trabalho foram cumpridos com sucesso, construindo um script conforme o solicitado, respeitando corretamente as opções de filtragem desejadas, e com visualização de mensagens de erro apropriadas para apoio ao utilizador.

Bibliografia

- Imagem de Capa adaptada de: https://www.hostinger.com.br/tutoriais/como-gerenciar-processos-no-linux-usando-linha-de-comando consultado a 24 de novembro de 2022
- https://stackoverflow.com/
- https://unix.stackexchange.com/
- https://www.computerhope.com/unix/bash/getopts.htm consultado a 15, 16 e 19 de novembro de 2022