Trabalho 1

Taxas de Leitura/Escrita de processos em bash

Licenciatura em Engenharia Informática

Sistemas Operativos

Docente:

Professor Nuno Lau

Alunos:

Bárbara Nóbrega Galiza – 105937

João Miguel Dias Andrade - 107969

Novembro de 2022

Índice

Introdução

Nesse trabalho, tivemos como objetivo o desenvolvimento de um “bash script” que obtém, a partir do ficheiro /proc/{pid}/io (em que {pid} corresponde ao id de um processo específico), estatísticas de leitura e escrita dos processos correntes no sistema, durante um período em segundos especificado pelo utilizador. O script, além de mostrar os bytes lidos/escritos e suas respetivas taxas, também informa o nome, pid, utilizador e data de início dos processos. A partir desses dados, são suportadas opções que permitem selecionar quais os processos a visualizar.

Ao longo deste relatório, será fornecida a explicação dos métodos utilizados para construir o código-fonte, seu fluxo de execução e o procedimento de testes realizados para validar o mesmo.

Abordagem

NOTA: nossa solução permite que o utilizador corra o programa com ou sem sudo, sendo que caso corra sem sudo os processos que exigem permissão não poderão ser visualizados.

Visão geral

A abordagem ao problema será apresentada em três blocos:

1) Leitura, armazenamento e manipulação de dados;

2) Processamento das opções e formatação da tabela;

3) Verificação dos argumentos.

Ao fazer essa divisão, buscamos separar os trechos de código por sua finalidade geral, e assim tornar nossa abordagem ao problema mais clara. Portanto, essa secção está dividida pela ordem lógica de construção da solução, e não pela ordem sequencial de execução do código.

1) Leitura, armazenamento e manipulação de dados

Primeira leitura

A fim de calcular o número de bytes lidos/escritos, fizemos dois ciclos **for,** com um **sleep** entre eles.

O primeiro ciclo percorre os processos contidos no ficheiro **/proc/** por meio do comando **ls** e de um **pipe** para o comando **grep[0-9]** para obter os processos identificados por dígitos (pids), ou seja, processos sem pid foram ignorados. Isso foi necessário porque processos sem pid não possuem a pasta **/io.** Dentro do ciclo, fizemos um **if** para verificar se a pasta **/proc/{pid}** existia, pois percebemos que havia processos que morriam antes de conseguirmos manipulá-los. Para isso, foi usado a opção –**d,** que informa se dado argumento é um diretório existente. Caso não exista mais, o ciclo avança para a próxima iteração através do **continue.** Em outro if, fizemos a verificação do sucesso do comando cat, visto que, caso o utilizador corra o programa sem sudo, muitos processos terão sua permissão negada e o cat retornará erro. Portanto, guardamos em uma variável temporária **rbf** o valor retornado pelo cat, e na mesma linha redirecionamos a saída do **stdout (2)** para o **/dev/null** para que as mensagens de erro não fossem

impressas no terminal. Depois, testamos se a saída do comando cat é 1 (insucesso): caso verdade, a instrução continue avança para a próxima iteração.

```
rbf=$(cat /proc/$pid/io 2>/dev/null)
if [[ $? == 1 ]]; then
    continue
fi
```

Para armazenar as linhas rchar e wchar do ficheiro **/proc/{pid}/io,** escolhemos usar um **vetor** como nossa única estrutura de dados, visto a funcionalidade da linguagem bash de guardar valores em um vetor a partir de um **índice,** sem a necessidade de inicializar aquele previamente. Assim, usamos os pids como índices de dois vetores: **rchar_before** e **wchar_before.** O primeiro guarda os caracteres lidos e o segundo os caracteres escritos, a partir de dois pipes e dos comandos **cat, sed** e **awk:**

```
rchar_before[$pid]=$(echo $rbf | sed -n 1p | awk '{print $2}')
wchar_before[$pid]=$(cat /proc/$pid/io | sed -n 2p | awk '{print $2}')
```

O cat concatena o conteúdo do ficheiro e retorna uma **stream** que vai através do pipe para o comando seguinte. O sed faz a seleção da linha e o awk a seleção da coluna, e assim o vetor fica, na posição especificada pelo pid, com o valor de caracteres lidos daquele processo.

Para o wchar_before, o processo é essencialmente o mesmo, só com uma diferença no número da linha a ser selecionada pelo sed (segunda linha).

Após concluirmos a primeira leitura, fizemos o sleep para esperar os segundos enviados como argumento do script. Para isso, buscamos o último argumento através do comando bash: `${@: -1}`.

Segunda leitura

Já no segundo ciclo for, percorremos os pids presentes no vetor rchar_before (que são os mesmos do wchar_before), utilizando a sintaxe bash que permite criar uma lista de todos os índices presentes em um vetor: `${!rchar_before[@]}`. Essa lista retorna apenas os índices em que foram guardados elementos, ou seja, o ciclo irá percorrer apenas os ids de processos que existiam na primeira leitura.

Assim como no primeiro ciclo, temos um if para verificar a existência do processo, com a diferença que aqui ele também assegura que os processos que existiam na primeira leitura, mas que morreram durante o sleep não sejam impressos. Em seguida, buscamos os dados dos processos para inserir na tabela: USER, DATE e COMM.

Para o user, fizemos ls para listar o próprio ficheiro, mas com a opção **l,** ou seja, no formato longo, no qual aparece a informação do utilizador daquele processo, combinado com um awk para selecionar a coluna respetiva.

Já para o campo date, separamos em duas fases: em primeiro lugar, buscamos a data de início do processo através do comando **ps** e opções **–p {pid}** e **–o lstart**. O comando ps lista processos e a opção p seleciona o processo a listar pelo pid. A opção -o lstart informa que a informação que queremos é a de data de início do processo. Como são impressas duas linhas, fizemos um pipe para o comando **tail –n1**, para buscar apenas a última, e terminamos com um awk que especifica as colunas onde se encontram o mês, dia e hora. Ainda dentro do awk, usamos a sintaxe de **substr** para obter apenas as horas e minutos, descartando os segundos, a fim de que a opção de especificação de período seja mais facilmente definida pelo utilizador. A escolha pelo comando ps ao invés do comando ls utilizado para o campo anterior se deu pelo facto de que o ls lista na língua da bash que executa o script, o que podia causar, e causou erros na bash configurada em português, devido aos nomes dos meses (por exemplo, Dez =/= Dec). Com o comando ps, os meses são listados na língua inglesa, o que padroniza o script e permite portabilidade.

Na segunda fase, transformamos a data para o formato de segundos corridos desde 1970 até agora, a partir do comando **date** com a formatação **%s** para fazer a conversão e a opção **d** para obter a data que especificamos ao invés da data atual. Guardamos o valor em uma nova variável, que vai ser utilizada futuramente para a comparação entre datas, no processamento das opções.

Finalmente, para o campo comm, criamos uma variável chamada **name** e guardamos nela o valor devolvido pelo cat do diretório **/proc/{pid}/comm.**

Após concluir a leitura dos campos, introduzimos o código responsável por ler novamente o número de caracteres lidos e escritos, idêntico ao utilizado na primeira leitura. Guardamos os valores em duas variáveis comuns (**rchar_after** e **wchar_after**), pois optamos por usar uma string que é incrementada a cada iteração com todos os campos de cada processo, e assim não houve necessidade de criar novos vetores. Com os novos valores guardados, fizemos a subtração dos rchar_after e wchar_after pelos valores contidos no vetor no índice de pid correspondente:

```
rchar=$((rchar_after-rchar_before[$pid]))
wchar=$((wchar_after-wchar_before[$pid]))
```

Como mencionado, usamos uma string para guardar os valores de cada processo. Isso foi feito usando o append presente no bash, **+=.** Assim, cada novo processo adiciona uma linha a essa string, com os campos correspondentes. Para os campos RATER e RATEW, que ainda não foram calculados, usamos a sintaxe **BEGIN** do awk para usar as variáveis rchar e wchar ao invés de um input. O BEGIN faz com que o awk execute a expressão antes de ler qualquer input, e nesse caso, como não há input, ele termina sua execução. Então, dividimos o valor de rchar e wchar pelo valor usado no sleep, ou seja, o valor do intervalo entre leituras, e obtemos as taxas de bytes lidos e escritos por segundo. `$(awk "BEGIN {print $rchar/${@: -1}}")`. Essa string foi colocada dentro de um if para que fossem concatenados apenas os processos especificados pelas opções, o que será discutido no tópico seguinte.

2) Processamento das opções e formatação da tabela

Para processar as opções, utilizamos o comando interno (builtin) do bash **getopts** em conjunto com um ciclo **while** e um **case**. Mas antes disso, tivemos que inicializar as variáveis que usaremos para guardar os argumentos das opções, pois caso o utilizador não especifique opções, então esses valores serão usados por defeito. Assim, para opções que utilizam **regex,** inicializamos variáveis com valores **'.*',** o que em regex significa "0 ou mais (*) de qualquer caractere (.)", ou seja, dá correspondência com qualquer string. Assim, se o utilizador não introduzir as opções c ou u, os processos impressos serão todos os que foram lidos.

Pela mesma lógica, inicializamos os valores de data mínima/máxima e pids mínimos/máximos com o menor e maior valor possível, respetivamente. No caso, o menor valor inteiro é 0, e o maior 2^{63}-1. Para o número de processos (opção -p), usamos um pipe do comando anteriormente utilizado para ler os processos, ls /proc/ | grep '[0-9]', para o comando **wc –l**, que imprime o número de linhas do ficheiro enviado como input, nesse caso, os processos. Multiplicamos esse número por 2, para garantir que processos criados entre o tempo decorrido entre esse ls e o ls presente no primeiro ciclo for também sejam impressos.

Para as opções restantes, nomeadamente w e r, criamos duas variáveis: **column** e **reverse.** A variável column define a coluna pela qual ordenaremos a tabela, e a variável reverse define se a ordem vai ser invertida ou não. Como por defeito queremos a tabela ordenada pela coluna RATER invertida, inicializamos column a 6 e reverse a 1.

Já com as variáveis inicializadas, podemos definir os novos valores das variáveis consoante a escolha das opções. No caso da opção r, mudamos o valor de reverse para 1. Já no caso da opção w, mudamos o valor de column para 7. Para as restantes opções, as variáveis recebem o valor de **${OPTARG},** ou seja, o valor introduzido como argumento referente à opção. No caso especial das variáveis **minimum_date** e **maximum_date,** foi necessário transformá-las para o formato de segundos corridos desde 1970, a fim de comparar com a data dos processos. Foi usado o mesmo método utilizado anteriormente:

```
minimum_date=$(date -d "${OPTARG}" +%s)
```

Como mencionado, a string com os valores dos processos foi encapsulada em um if para testar as opções, nomeadamente as opções que requerem argumentos (com exceção da opção p). Para comparar os regex, usamos a sintaxe **~=,** que verifica se a string da esquerda dá match com o regex à direita. Para as datas, comparamos a data introduzida em segundos com a data do início do processo também em segundos, e para os pids, fizemos uma comparação direta, sendo que em ambas usamos comparadores aritméticos como –ge (greater or equal) e –le (less or equal). Caso uma das opções falhe, o processo não é concatenado à string (instruções &&).

Para as opções r e w, fizemos um novo bloco if/else, agora fora do ciclo for. Caso reverse = 1, ordenamos a string em ordem descrescente com o comando **sort –gr.** Caso reverse = 0, ordenamos a string em ordem crescente com o comando **sort –g.** A opção g é usada para comparar corretamente valores números com exponenciais, caso estes apareçam, o que não é suportado pela opção n. Além disso, em ambas as opções também usamos as opções **t,** para separar as colunas por ";", e **k,** para especificar por qual coluna queremos ordenar, a partir da variável column. Em seguida, usamos um pipe para selecionar o número de linhas, ou seja, o número de processos (opção p), através do comando **head –n** com o argumento correspondente ao valor da variável definida anteriormente no case da opção p (ou por defeito). As somas correspondem ao incremento necessário para a formatação correta.

```
if [[ $reverse -eq 1 ]];then
  format=$(echo -e "$format" | sort -gr -t ";" -k $column,$column)
  echo -e "COMM;USER;PID;READB;WRITEB;RATER;RATEW;DATE\n$format" | head -n $(($lines+1)) | column -s ";" -t
else
  format=$(echo -e "$format" | sort -g -t ";" -k $column,$column)
  echo -e "COMM;USER;PID;READB;WRITEB;RATER;RATEW;DATE$format" | head -n $(($lines+1)) | column -s ";" -t
fi
```

Finalmente, fizemos echo do cabeçalho seguido da string format com um pipe para criar a tabela baseado no delimitador ";", usando o comando **column**. Escolhemos esse separador para evitar problemas com números decimais, que podem usar pontos ou vírgulas, e problemas com o nome de processos, que poderiam ter '-', ':', etc, mas que nunca têm ";".

3) Verificação dos argumentos

Para verificar os argumentos das opções e o argumento final obrigatório, utilizamos alguns if's, uma função e a opção default '?' no case. Em todas as verificações, caso a condição testada for verdadeira, o programa imprime uma mensagem de erro e termina a execução com o status de erro 1 (**exit 1**).

Para verificar a introdução do argumento final (número de segundos entre leituras), testamos em um if se a quantidade de argumentos era menor que 1, utilizando **$#**. Além disso, para verificar se o argumento é válido, isto é, se é um número inteiro positivo, utilizamos um if com regex e comparação aritmética.

No caso dos argumentos das opções numéricos, utilizamos do mesmo if usado para validar o último argumento, só que encapsulado em uma função, para que fosse reutilizado em cada opção do case que aceitasse esses argumentos, nomeadamente as opções m, M e p.

```
function check_arg_is_num(){
    if ! [[ $1 =~ ^[0-9]+$ && $1 > 0 ]] ; then   # verificar se o argumento é válido
      echo "Erro. Argumento deve ser um número inteiro positivo." >&2
      exit 1
    fi
}
```

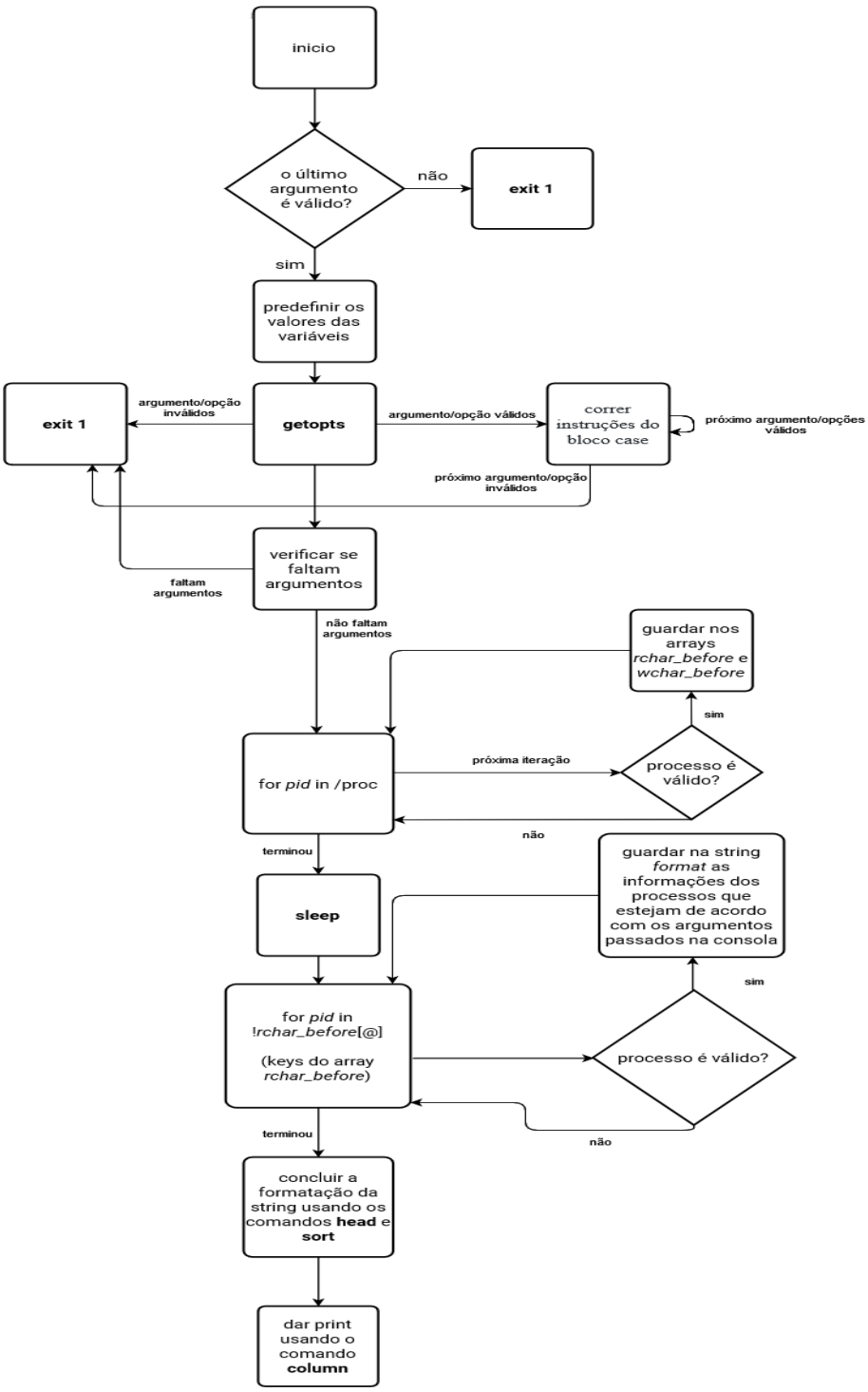
Assim, em cada opção no case, a função é chamada com o argumento presente em {$OPTARG}.

Para as datas, utilizamos um if em cada bloco do case, em que se o valor de retorno do último comando (**$?**) for igual a 1 (erro), termina o programa. No caso o comando testado é o date, que gera um erro se seu argumento for uma data inválida.

Já no caso em que o utilizador introduz uma opção inválida, utilizamos a sintaxe '?)' do case para imprimir uma mensagem de erro, informando as opções existentes, e terminar o programa. Vale ressaltar que adicionamos ":" ao início da optstring a fim de fazer um "silent error checking", ou seja, fazer com que o getopts não imprima uma mensagem de erro, visto que já definimos nossa própria mensagem de erro.

Fora do getopts, fizemos uma última verificação que compara o número de argumentos $# com o valor da variável **${OPTIND}.** Isso foi feito a fim de impedir que o utilizador introduzisse alguma das opções sem argumentos. O valor de ${OPTIND} é incrementado a cada processamento de opção, e aponta para a posição da próxima opção a ser lida. Se a chamada ao programa for válida, após processar todas as opções, o valor de ${OPTIND} deverá corresponder a posição do último argumento, ou seja, ao número de argumentos devolvido por $# (bash não considera o executável para a contagem de número de argumentos, por isso $# = posição do último argumento).

Fluxo de Execução



Testes

Nesta secção serão apresentados os testes realizados para validar a nossa solução. Cada teste seguirá um padrão no qual definimos:

1) O objetivo;

2) O teste;

3) O resultado;

4) A conclusão;

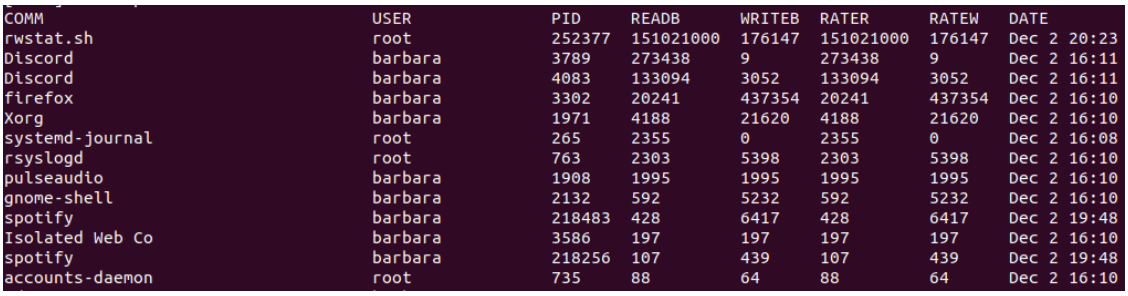
TESTE 1

Objetivo: Confirmar que o programa funciona com sudo e que está ordenado pela ordem inversa da taxa de leitura por defeito.

Teste:

```
$ sudo ./rwstat.sh 1
```

Resultado (imagem cortada mostra primeiras linhas):



COMM	USER	PID	READB	WRITEB	RATER	RATEW	DATE
rwstat.sh	root	252377	151021000	176147	151021000	176147	Dec 2 20:23
Discord	barbara	3789	273438	9	273438	9	Dec 2 16:11
Discord	barbara	4083	133094	3052	133094	3052	Dec 2 16:11
firefox	barbara	3302	20241	437354	20241	437354	Dec 2 16:10
Xorg	barbara	1971	4188	21620	4188	21620	Dec 2 16:10
systemd-journal	root	265	2355	0	2355	0	Dec 2 16:08
rsyslogd	root	763	2303	5398	2303	5398	Dec 2 16:10
pulseaudio	barbara	1908	1995	1995	1995	1995	Dec 2 16:10
gnome-shell	barbara	2132	592	5232	592	5232	Dec 2 16:10
spotify	barbara	218483	428	6417	428	6417	Dec 2 19:48
Isolated Web Co	barbara	3586	197	197	197	197	Dec 2 16:10
spotify	barbara	218256	107	439	107	439	Dec 2 19:48
accounts-daemon	root	735	88	64	88	64	Dec 2 16:10

Conclusão: Como aparecem processos da root e do utilizador, confirmamos que o programa teve acesso a ficheiros que requerem permissão a partir do sudo. Além disso, como os processos foram apresentados por ordem decrescente da taxa de leitura, confirmamos o pressuposto.

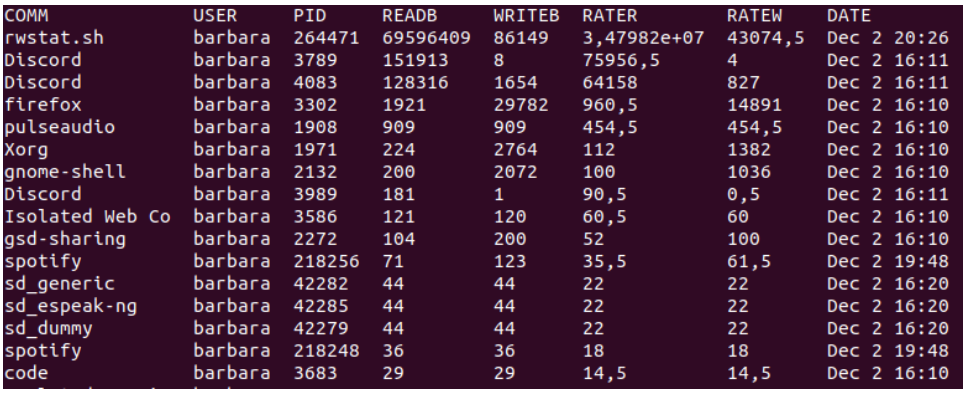
TESTE 2

Objetivo: Confirmar que o programa funciona sem sudo, e que as taxas estão a ser calculadas corretamente.

Teste:

```
./rwstat.sh 2
```

Resultado (imagem cortada mostra primeiras linhas):



COMM	USER	PID	READB	WRITEB	RATER	RATEW	DATE
rwstat.sh	barbara	264471	69596409	86149	3,47982e+07	43074,5	Dec 2 20:26
Discord	barbara	3789	151913	8	75956,5	4	Dec 2 16:11
Discord	barbara	4083	128316	1654	64158	827	Dec 2 16:11
firefox	barbara	3302	1921	29782	960,5	14891	Dec 2 16:10
pulseaudio	barbara	1908	909	909	454,5	454,5	Dec 2 16:10
Xorg	barbara	1971	224	2764	112	1382	Dec 2 16:10
gnome-shell	barbara	2132	200	2072	100	1036	Dec 2 16:10
Discord	barbara	3989	181	1	90,5	0,5	Dec 2 16:11
Isolated Web Co	barbara	3586	121	120	60,5	60	Dec 2 16:10
gsd-sharing	barbara	2272	104	200	52	100	Dec 2 16:10
spotify	barbara	218256	71	123	35,5	61,5	Dec 2 19:48
sd_generic	barbara	42282	44	44	22	22	Dec 2 16:20
sd_espeak-ng	barbara	42285	44	44	22	22	Dec 2 16:20
sd_dummy	barbara	42279	44	44	22	22	Dec 2 16:20
spotify	barbara	218248	36	36	18	18	Dec 2 19:48
code	barbara	3683	29	29	14,5	14,5	Dec 2 16:10

Conclusão: Como só aparecem processos do utilizador, confirmamos que o programa não teve acesso a ficheiros que requerem permissão, mas que mesmo assim correu sem erros. Em adição, como os valores das taxas são metade do número de bytes lidos/escritos, concluímos que o cálculo está correto.

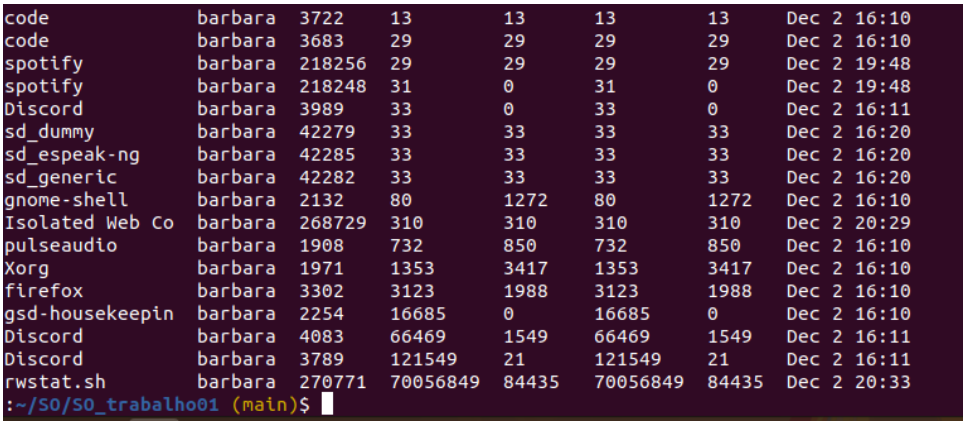
TESTE 3

Objetivo: Assegurar o funcionamento correto da opção r.

Teste:

```
$ ./rwstat.sh -r 1
```

Resultado (imagem cortada mostra últimas linhas):



code	barbara	3722	13	13	13	13	Dec 2 16:10
code	barbara	3683	29	29	29	29	Dec 2 16:10
spotify	barbara	218256	29	29	29	29	Dec 2 19:48
spotify	barbara	218248	31	0	31	0	Dec 2 19:48
Discord	barbara	3989	33	0	33	0	Dec 2 16:11
sd_dummy	barbara	42279	33	33	33	33	Dec 2 16:20
sd_espeak-ng	barbara	42285	33	33	33	33	Dec 2 16:20
sd_generic	barbara	42282	33	33	33	33	Dec 2 16:20
gnome-shell	barbara	2132	80	1272	80	1272	Dec 2 16:10
Isolated Web Co	barbara	268729	310	310	310	310	Dec 2 20:29
pulseaudio	barbara	1908	732	850	732	850	Dec 2 16:10
Xorg	barbara	1971	1353	3417	1353	3417	Dec 2 16:10
firefox	barbara	3302	3123	1988	3123	1988	Dec 2 16:10
gsd-housekeepin	barbara	2254	16685	0	16685	0	Dec 2 16:10
Discord	barbara	4083	66469	1549	66469	1549	Dec 2 16:11
Discord	barbara	3789	121549	21	121549	21	Dec 2 16:11
rwstat.sh	barbara	270771	70056849	84435	70056849	84435	Dec 2 20:33

```
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como os processos estão ordenados por ordem crescente da taxa de leitura (coluna 6), concluímos que a opção está a funcionar corretamente.

TESTE 4

Objetivo: Assegurar o funcionamento correto da opção w.

Teste:

```
$ ./rwstat.sh -w 1
```

Resultado (imagem cortada mostra primeiras linhas):

```
:~/SO/SO_trabalho01 (main)$ ./rwstat.sh -w 1
COMM             USER     PID     READB     WRITEB  RATER     RATEW   DATE
gnome-terminal-  barbara  4577    16        175156  16        175156  Dec 2 16:14
rwstat.sh        barbara  275011  70979484  85118   70979484  85118   Dec 2 20:38
Xorg             barbara  1971    8206      48430   8206      48430   Dec 2 16:10
gnome-shell      barbara  2132    1664      15748   1664      15748   Dec 2 16:10
firefox          barbara  3302    1372      13001   1372      13001   Dec 2 16:10
spotify          barbara  218483  388       6417    388       6417    Dec 2 19:48
code             barbara  3721    4991      4322    4991      4322    Dec 2 16:10
code             barbara  3650    121       3185    121       3185    Dec 2 16:10
Discord          barbara  4083    947       1173    947       1173    Dec 2 16:11
Isolated Web Co  barbara  268729  1115      1116    1115      1116    Dec 2 20:29
pulseaudio       barbara  1908    555       555     555       555     Dec 2 16:10
code             barbara  3566    283       495     283       495     Dec 2 16:10
code             barbara  3683    283       483     283       483     Dec 2 16:10
spotify          barbara  218256  78        358     78        358     Dec 2 19:48
```

Conclusão: Como os processos estão ordenados por ordem inversa da taxa de escrita, concluímos que a opção está a funcionar corretamente.

TESTE 5

Objetivo: Assegurar o funcionamento correto da opção r em conjunto da opção w.

Teste:

```
$ ./rwstat.sh -w -r 1
```

Resultado (imagem cortada mostra últimas linhas):

```
code             barbara  3722    13        13      13        13      Dec 2 16:10
Privileged Cont  barbara  3411    14        14      14        14      Dec 2 16:10
Isolated Web Co  barbara  268656  19        19      19        19      Dec 2 20:29
Discord          barbara  3789    121549    21      121549    21      Dec 2 16:11
code             barbara  3683    29        29      29        29      Dec 2 16:10
spotify          barbara  218256  30        30      30        30      Dec 2 19:48
code             barbara  3749    0         31      0         31      Dec 2 16:11
sd_dummy         barbara  42279   33        33      33        33      Dec 2 16:20
sd_espeak-ng     barbara  42285   33        33      33        33      Dec 2 16:20
sd_generic       barbara  42282   33        33      33        33      Dec 2 16:20
firefox          barbara  3302    843       89      843       89      Dec 2 16:10
Isolated Web Co  barbara  268729  167       167     167       167     Dec 2 20:29
pulseaudio       barbara  1908    606       606     606       606     Dec 2 16:10
Discord          barbara  4083    876       1328    876       1328    Dec 2 16:11
gnome-shell      barbara  2132    464       3288    464       3288    Dec 2 16:10
gnome-terminal-  barbara  4577    0         3612    0         3612    Dec 2 16:14
Xorg             barbara  1971    2020      9248    2020      9248    Dec 2 16:10
rwstat.sh        barbara  279184  70385875  84456   70385875  84456   Dec 2 20:40
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como os processos estão ordenados por ordem crescente da taxa de escrita (coluna 7), concluímos que as opções estão a funcionar corretamente.

TESTE 6

Objetivo: Assegurar o funcionamento correto da opção c, ao utilizar um regex que dá match com strings iniciadas com a letra d.

Teste:

```
$ ./rwstat.sh -c "^d" 1
```

Resultado:

```
:~/SO/SO_trabalho01 (main)$ ./rwstat.sh -c "^d" 1
COMM            USER     PID   READB  WRITEB  RATER  RATEW  DATE
duplicity       barbara  4713  0      0       0      0      Dec 2 16:14
deja-dup-monito barbara  4515  0      0       0      0      Dec 2 16:12
deja-dup        barbara  4607  0      0       0      0      Dec 2 16:14
dconf-service   barbara  2204  0      0       0      0      Dec 2 16:10
dbus-daemon     barbara  2103  0      0       0      0      Dec 2 16:10
dbus-daemon     barbara  1913  0      0       0      0      Dec 2 16:10
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como os processos que aparecem começam pela letra d, asseguramos que a opção c está a funcionar corretamente.

TESTE 7

Objetivo: Assegurar o funcionamento correto da opção u, ao utilizar um regex que dá match com strings iniciadas com a letra r, e assegurar o funcionamento da opção p. Aqui utilizamos sudo para efeitos de comparação.

Teste:

```
$ sudo ./rwstat.sh -u "^r" -p 8 1
```

Resultado:

```
COMM            USER  PID     READB      WRITEB  RATER      RATEW   DATE
rwstat.sh       root  535060  149221546  135633  149221546  135633  Dec 2 21:25
irqbalance      root  754     4567       0       4567       0       Dec 2 16:10
thermald        root  777     29         0       29         0       Dec 2 16:10
systemd-journal root  265     8          0       8          0       Dec 2 16:08
rtkit-daemon    root  1052    8          8       8          8       Dec 2 16:10
containerd      root  846     2          2       2          2       Dec 2 16:10
zswap-shrink    root  130     0          0       0          0       Dec 2 16:08
writeback       root  42      0          0       0          0       Dec 2 16:08
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como os processos que aparecem tem users cujo nome começa pela letra r, asseguramos que a opção u está a funcionar corretamente. Além disso, conferimos que só aparecem 8 processos, como especificado pela opção p.

TESTE 8

Objetivo: Assegurar o funcionamento correto das opções ‘s’ e ‘e’.

Teste:

```
$ ./rwstat.sh -s "Dec 2 19:00" -e "Dec 2 21:00" 1
```

Resultado:

```
COMM             USER     PID     READB  WRITEB  RATER  RATEW  DATE
Isolated Web Co  barbara  268729  81     81      81     81     Dec 2 20:29
spotify          barbara  218484  41     1       41     1      Dec 2 19:48
spotify          barbara  218483  41     1       41     1      Dec 2 19:48
spotify          barbara  218256  28     28      28     28     Dec 2 19:48
spotify          barbara  218431  5      0       5      0      Dec 2 19:48
spotify          barbara  218429  5      0       5      0      Dec 2 19:48
Isolated Web Co  barbara  268772  4      4       4      4      Dec 2 20:29
Isolated Web Co  barbara  268703  4      4       4      4      Dec 2 20:29
spotify          barbara  218248  3      3       3      3      Dec 2 19:48
Isolated Web Co  barbara  268656  2      2       2      2      Dec 2 20:29
spotify          barbara  218551  0      0       0      0      Dec 2 19:48
spotify          barbara  218511  0      0       0      0      Dec 2 19:48
spotify          barbara  218478  0      0       0      0      Dec 2 19:48
spotify          barbara  218477  0      0       0      0      Dec 2 19:48
spotify          barbara  218401  0      0       0      0      Dec 2 19:48
spotify          barbara  218400  0      0       0      0      Dec 2 19:48
spotify          barbara  218399  0      0       0      0      Dec 2 19:48
spotify          barbara  218398  0      0       0      0      Dec 2 19:48
Terminal
:~/SO/SO_[illegible]01 (main)$
```

Conclusão: Como os processos que aparecem tem datas dentro da gama especificada, concluímos que as opções estão a funcionar corretamente.

TESTE 9

Objetivo: Assegurar o funcionamento correto das opções m e M.

Teste:

```
$ ./rwstat.sh -m 1000 -M 2000 1
```

Resultado:

```
COMM            USER     PID   READB  WRITEB  RATER  RATEW  DATE
pulseaudio      barbara  1908  606    606     606    606    Dec 2 16:10
Xorg            barbara  1971  169    1997    169    1997   Dec 2 16:10
tracker-miner-f barbara  1910  0      0       0      0      Dec 2 16:10
systemd         barbara  1902  0      0       0      0      Dec 2 16:10
gvfs-udisks2-vo barbara  1945  0      0       0      0      Dec 2 16:10
gvfs-mtp-volume barbara  1985  0      0       0      0      Dec 2 16:10
gvfs-gphoto2-vo barbara  1958  0      0       0      0      Dec 2 16:10
gvfs-goa-volume barbara  1962  0      0       0      0      Dec 2 16:10
gvfsd-fuse      barbara  1938  0      0       0      0      Dec 2 16:10
gvfsd           barbara  1933  0      0       0      0      Dec 2 16:10
gvfs-afc-volume barbara  1953  0      0       0      0      Dec 2 16:10
goa-identity-se barbara  1979  0      0       0      0      Dec 2 16:10
goa-daemon      barbara  1967  0      0       0      0      Dec 2 16:10
gdm-x-session   barbara  1969  0      0       0      0      Dec 2 16:10
dbus-daemon     barbara  1913  0      0       0      0      Dec 2 16:10
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como os processos que aparecem tem pids dentro da gama especificada, concluímos que as opções estão a funcionar corretamente.

TESTE 10

Objetivo: Assegurar a validação do número de argumentos, ao omitir o argumento obrigatório.

Teste:

```
$ ./rwstat.sh -m 1000 -M 2000
```

Resultado:

```
Erro, faltam argumentos.
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como o programa imprime uma mensagem de erro e termina, concluímos que a validação funcionou.

TESTE 11

Objetivo: Assegurar a validação do número de argumentos, ao omitir o argumento da opção c.

Teste:

```
$ ./rwstat.sh -c -p 5 2
```

Resultado:

```
Erro, faltam argumentos.
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como o programa imprime uma mensagem de erro e termina, concluímos que a validação funcionou.

TESTE 12

Objetivo: Assegurar a validação do número de argumentos (introdução do argumento obrigatório).

Teste:

```
$ ./rwstat.sh
```

Resultado:

```
Erro. Indique o número de segundos que serão usados para calcular as taxas de I/O.
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como o programa imprime uma mensagem de erro e termina, concluímos que a validação funcionou.

TESTE 13

Objetivo: Assegurar a validação do último argumento.

Teste:

```
$ ./rwstat.sh -2
```

Resultado:

```
Erro. O último argumento tem de ser um inteiro positivo.
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como o programa imprime uma mensagem de erro e termina, concluímos que a validação funcionou.

TESTE 14

Objetivo: Assegurar a validação dos argumentos numéricos.

Teste:

```
$ ./rwstat.sh -p -1 1
```

Resultado:

```
Erro. Argumento deve ser um número inteiro positivo.
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Como o programa imprime uma mensagem de erro e termina, concluímos que a validação funcionou.

TESTE 15

Objetivo: Assegurar o funcionamento de várias opções em conjunto. O regex da opção c deverá retornar apenas processos que terminem com a letra e.

Teste:

```
$ ./rwstat.sh -c "e$" -r -w -m 1000 1
```

Resultado:

```
COMM             USER     PID   READB  WRITEB  RATER  RATEW  DATE
code             barbara  3112  0      0       0      0      Dec 2 22:46
code             barbara  3145  0      0       0      0      Dec 2 22:46
code             barbara  3146  0      0       0      0      Dec 2 22:46
code             barbara  3148  0      0       0      0      Dec 2 22:46
code             barbara  3174  0      0       0      0      Dec 2 22:46
code             barbara  3184  0      0       0      0      Dec 2 22:46
code             barbara  3317  13     0       13     0      Dec 2 22:46
code             barbara  3421  0      0       0      0      Dec 2 22:46
dconf-service    barbara  2192  0      0       0      0      Dec 2 22:46
evolution-addre  barbara  2202  0      0       0      0      Dec 2 22:46
goa-identity-se  barbara  1965  0      0       0      0      Dec 2 22:46
gsd-datetime     barbara  2240  0      0       0      0      Dec 2 22:46
gvfs-afc-volume  barbara  1941  0      0       0      0      Dec 2 22:46
gvfsd-fuse       barbara  1926  0      0       0      0      Dec 2 22:46
gvfs-goa-volume  barbara  1950  0      0       0      0      Dec 2 22:46
gvfs-mtp-volume  barbara  1973  0      0       0      0      Dec 2 22:46
snap-store       barbara  2377  0      0       0      0      Dec 2 22:46
code             barbara  3316  1490   8       1490   8      Dec 2 22:46
code             barbara  3202  2968   21      2968   21     Dec 2 22:46
code             barbara  3346  0      31      0      31     Dec 2 22:46
:~/SO/SO_trabalho01 (main)$
```

Conclusão: Todas as opções funcionaram, visto que só foram impressos processos cujo nome termina em ‘e’, com pids maiores que 1000 e na ordem crescente de taxa de escrita.

Conclusão

Esse trabalho nos permitiu aprender mais sobre a sintaxe bash e o poder que esta linguagem de scripting possui. Aprendemos a tornar a execução do script personalizável com as opções, o que o deixa muito mais amigável ao utilizador, além de permitir uma melhor visualização dos processos correntes no sistema operativo, no caso do código em questão.

Além disso, nos familiarizamos com o ficheiro /proc/, com os pids, usuários, formatos de datas, especificações da bash e com linguagens extremamente úteis como é o caso da AWK.

Finalmente, concluímos que esse trabalho contribuiu imenso para o nosso conhecimento acerca do funcionamento do Unix e para o melhor aproveitamento dos recursos dos oferecidos por esse sistema operativo.

Bibliografia

Guiões práticos da disciplina

https://www.computerhope.com/unix/bash/getopts.htm

https://stackoverflow.com/questions/14249931/how-does-the-optind-variable-work-in-the-shell-builtin-getopts

https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html

https://www.gnu.org/software/gawk/manual/gawk.html

https://opensource.com/article/18/5/you-dont-know-bash-intro-bash-arrays

https://www.cyberciti.biz/faq/linux-unix-bsd-apple-osx-bash-get-last-argument/