Relatório do Trabalho 01 de Sistemas Operativos 2021/2022

Trabalho realizado por: André Butuc (Nmec: 103530);
Gonçalo Silva (Nmec: 103668).

Data da última alteração: 4/12/2021.

Introdução

Com este relatório visamos fornecer uma explicação sucinta do código elaborado, submetido com o nome "netifstat.sh", um script em bash que permite a visualização da quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas e as respetivas taxas de transferência para períodos de tempo pré-estabelecidos.

O relatório irá abordar as metodologias de programação que utilizámos, quer na utilização de estrutura de dados, algoritmos de ordenação, funções internas ao bash e métodos de validação dos argumentos passados na linha de comandos.

Estrutura do relatório:

I. Estrutura do programa

- Recolha dos nomes das interfaces de rede (linha 4 – linha 10)
- Inicialização das variáveis necessárias para as flags e para a validação e controlo dos argumentos passados na linha de comandos (linha 36 – linha 61 | l linha 291 – linha 292 | linha 366)
- Processamento e controlo dos argumentos da linha de comandos (linha 13 – linha 33 | linha 62 – linha 247)
- Inicialização das listas necessárias (linha 250 – linha 288)
- Recolha dos dados associadas a cada interface de rede (linha 296 – linha 302 | linha 307 – linha 313)
- Sleep durante os segundos introduzidos na linha de comandos (linha 305)
- Cálculo dos dados necessários a partir dos dados obtidos (linha 315 – linha 325)
- Conversão dos dados em bytes, kilobytes ou megabytes (linha 329 – linha 352)

- Cálculo das taxas de transferência (linha 356 – linha 361)
- Ordenação dos dados tendo em consideração a flag de ordenação ativa (linha 367 – linha 432)
- Impressão dos dados (linha 435 – linha 470)

II. Estruturas de dados utilizadas e a lógica de organização de dados

Em todo o código foi utilizada somente uma estrutura de dados: as listas.
As listas presentes no programa são as seguintes:

a) interfaces – contém os nomes das interfaces.
b) rx_bytes_i – contém os dados relativos aos rx bytes iniciais, ou seja, os bytes que se já se encontram recebidos na primeira chamada ao sistema.
c) rx_bytes_f – contém os dados relativos aos rx bytes finais, ou seja, os bytes que após "x" segundos (sendo "x" o argumento passado na linha de comandos relativo ao número de segundos que serão usados para calcular as taxas de transferência) se encontram como recebidos na mais recente chamada ao sistema.
d) tx_bytes_i – contém que contém os dados relativos aos tx bytes iniciais, ou seja, os bytes que se já se encontram transmitidos na primeira chamada ao sistema.
e) tx_bytes_f – contém os dados relativos aos tx bytes finais, ou seja, os bytes que após "x" segundos (sendo "x" o argumento passado na linha de comandos relativo ao número de segundos que serão usados para calcular as taxas de transferência) se encontram como transmitidos na mais recente chamada ao sistema.
f) rx_bytes – contém os bytes que consistem no valor da diferença entre os elementos correspondentes das listas "rx_bytes_f" e "rx_bytes_i";
g) tx_bytes – contém os bytes que consistem no valor da diferença entre os elementos correspondentes das listas "tx_bytes_f" e "tx_bytes_i";
h) rrate – contém os bytes que consistem no valor da divisão (com vírgula flutuante) entre os elementos da lista "rx_bytes" e o número de segundos passados como argumento na linha de comandos;
i) trate – contém os bytes que consistem no valor da divisão (com vírgula flutuante) entre os elementos da lista "tx_bytes" e o número de segundos passados como argumento na linha de comandos;
j) ordem – contém os valores dos índices da lista "interfaces", será uma lista essencial no processo de ordenação dos dados, sendo também utilizada na impressão dos dados no terminal;
k) temp_ordem – contém os valores dos índices da lista "interfaces", mas ordenados inversamente relativamente à lista "ordem";
l) total_rx_bytes – contém os valores acumulados da lista "rx_bytes" em cada iteração do loop infinito (acionado pela flag "-l")
m) total_tx_bytes – contém os valores acumulados da lista "tx_bytes" em cada iteração do loop infinito (acionado pela flag "-l")

Todos os elementos das listas são inicializados com o valor 0 através de um for loop que itera o número de interfaces vezes, à exceção da lista "interfaces" e "ordem". A

lista “interfaces” não é inicializada, mas é fruto da concatenação dos sucessivos nomes das interfaces de rede. A lista “ordem” é inicializada com os índices da lista “interfaces” por ordem crescente (padrão).

Para uma fácil organização dos dados criámos um sistema de índices fixos que nos dá as informações de cada interface. Nas listas “rx_bytes_i”, “rx_bytes_f”, “tx_bytes_i”, “tx_bytes_f”, “rx_bytes”, “tx_bytes”, “rrate”, “trate”, “total_rx_bytes”, “total_tx_bytes” os valores que se encontram nos índices 0, 1, 2, correspondem às interfaces que estão nos índices 0, 1, 2 na lista “interfaces”. Ou seja, o valor rx_bytes_i[0] e rx_bytes_f[0] são referentes à interface que está no índice 0 da lista “interfaces” e assim sucessivamente. A ordem de cada lista mencionada neste sistema é fixa e assegurada durante a execução do script.

III. Processamento e validação dos argumentos passados na linha de comandos

Para o processamento dos argumentos passados na linha de comandos utilizámos variáveis para armazenar os valores 0 ou 1 referentes às flags passadas (0 – flag desativada; 1 – flag ativada) e os conteúdos que precedem certas flags (a flag “-c” e a flag “-p”). Exceção: variável “time” tem sempre o último argumento passado na linha de comandos.

Variáveis:

a) regex – armazena a expressão regular passada após a flag “-c”;
b) byte – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-b”;
c) kbyte – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-k”;
d) mbyte – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-m”;
e) n_interfaces_display – armazena o valor padrão número de elementos da lista “interfaces”, caso seja passada a flag “-p” é armazenado o valor que sucede a flag;
f) sort_TX – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-t”;
g) sort_RX – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-r”;
h) sort_TRATE – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-T”;
i) sort_RRATE – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-R”;
j) reverse – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-v”;
k) loop – armazena o valor padrão de 0 ou 1, caso seja passada a flag “-l”;
l) time – armazena o valor que é passado como último argumento na linha de comandos.

Para ativar cada flag e fazer o seu processamento foi utilizada a função “getopts” com a seguinte string como argumento “:c: :b :k :m p: :t :r :T :R :v :l “, como a flag “-c” precede uma expressão regular colocámos “:” à frente para o “getopts” reconhecer que será necessário guardar o valor que precede a flag na variável “OPTARG”, o mesmo acontece com a flag “-p” que precede um valor inteiro que corresponde ao número de interfaces que serão imprimidas. Os “:” atrás de cada flag, permite-nos controlar o caso de ser passado um argumento fora das flags tabeladas, sendo, nesta

situação, corrido o código situado no default "\?", que imprime uma mensagem de erro e termina o programa com "exit code" igual a um.

Para obter o conteúdo da variável "time", utilizámos a indexação negativa do array "@" para obter o seu último elemento (que é sempre o valor de segundos utilizados para as taxas).

Relativamente ao controlo dos argumentos passados na linha de comandos foram implementadas estruturas de condição (estrutura if-then) para não permitir que seja possível serem ativadas múltiplas flags do mesmo tipo. Ou seja, dentro das flags de ordenação, só pode estar ativa uma ("-b", "-k" ou "-m"), caso esteja mais do que uma ativa, é impressa uma mensagem de erro e o programa é terminado com "exit code" igual a um. O mesmo se aplica às flags de conversão, sendo que só pode estar ativa uma ("-t", "-r", "-T" ou "-R"), caso esteja mais do que uma ativa, é impressa uma mensagem de erro e o programa é terminado com "exit code" igual a um.

Ainda no assunto do controlo dos argumentos, também é assegurado que o valor da variável "OPTARG" correspondente à flag "-p" é um valor inteiro menor ou igual ao número de interfaces disponíveis. Caso contrário, é impressa uma mensagem de erro e o programa termina com "exit code" igual a um.

Para tornar o programa ainda mais consistente e seguro implementámos também as seguintes variáveis:

a) c_flag – inicialmente com o valor padrão de 0, quando a flag "-c" é ativada, o seu conteúdo muda para o valor de 1;
b) p_flag - inicialmente com o valor padrão de 0, quando a flag "-p" é ativada, o seu conteúdo muda para o valor de 1;
c) c_count – variável que conta o número de vezes que a flag "-c" é ativada;
d) b_count – variável que conta o número de vezes que a flag "-b" é ativada;
e) k_count – variável que conta o número de vezes que a flag "-k" é ativada;
f) m_count – variável que conta o número de vezes que a flag "-m" é ativada;
g) p_count – variável que conta o número de vezes que a flag "-p" é ativada;
h) t_count – variável que conta o número de vezes que a flag "-t" é ativada;
i) r_count – variável que conta o número de vezes que a flag "-r" é ativada;
j) T_count – variável que conta o número de vezes que a flag "-T" é ativada;
k) R_count – variável que conta o número de vezes que a flag "-R" é ativada;
l) v_count – variável que conta o número de vezes que a flag "-v" é ativada;
m) l_count – variável que conta o número de vezes que a flag "-l" é ativada;
n) last_processed – variável que regista se a última flag processada foi "-c" ou "-p" registando o valor de 1 caso foram, caso contrário regista o valor 0;
o) second_to_last_argument – variável que regista o penúltimo argumento passado na linha de comandos;

As variáveis "c_flag" e "p_flag", permitem controlar o número de inteiros que são passados na linha de comandos, sendo que estes, excluindo o argumento do tempo, só poderão suceder a flag "-c", sendo interpretado como uma expressão regular, ou

então a flag “-p”, sendo interpretado como o número de interfaces a mostrar. Usando a variável “possible_int_flags” para guardar a soma de “c_flag” com “p_flag” e 1 (o argumento obrigatório dos segundos) avaliamos se o número de argumentos inteiros passos é superior ao valor de “possible_int_flags”, caso seja, é impressa uma mensagem de erro e o programa termina com “exit code” igual a um.

As variáveis “c_count”, “b_count”, “k_count”, “m_count”, “p_count”, “t_count”, “r_count”, “T_count”, “R_count”, “v_count”, “l_count” são usadas para controlar se uma flag não é passada mais do que uma vez. Caso seja, é impressa uma mensagem de erro e o programa termina com “exit code” igual a um.

A variável “last_processed” é utilizada no controlo dos casos em que antes de uma flag ou depois é passado um caractere ou inteiro não expectável o que faz com o que o getopts interrompa o processamento das flags. Imaginemos que é passado o seguinte na linha de comandos “./netifstat.sh -c d 2 2 2”, o valor da variável do getopts “OPTIND” fica com o valor de 3 que iria ser o próximo argumento lido pelo getopts, mas o processamento foi interrompido porque “2” não é uma flag. Numa situação de sucesso o “OPTIND” fica com o valor de “#”, mas, mesmo com sucesso, se o última flag a processar for “-c” ou “-p” o “OPTIND” será igual a “#”-1. Logo, caso o “OPTIND” tenha um valor inferior aos mencionados anteriormente, ocorreu um “conflito” devido aos argumentos passados na linha de comandos.

A variável “second_to_last_argument” é utilizada para verificar se o penúltimo argumento é “-p” ou “-c”, pois caso seja, irá criar um conflito com o valor do tempo. Este valor seria considerado tanto para a variável “time” como para a variável “n_interfaces_display” (caso a flag “-p” seja ativa) ou para a variável “regex” (caso a flag “-c” seja ativa), o que facto é um caso incorreto e inválido.

IV. Obtenção dos dados relativos às interfaces

Para obter os nomes das interfaces de rede do computador conjugamos as funções “ifconfig”, “cut”, “tr”.

A função “ifconfig” imprime as informações relativas a cada interface de rede.

A função “cut” com delimitador “ “ e flag “-f 1”, sendo “1” o “field number” extrai do “output” do “ifconfig” a primeira coluna, que contém os nomes das interfaces. Por exemplo, caso as nossas interfaces de rede fossem “enp7s0” e “lo”, o código “cut -d “ “ -f1” nos daria o seguinte output: “enp7s0: lo:”.

A função “tr” com os argumentos “:” e “\n” vai remover os caracteres passados como argumento do output da função “cut”. Utilizando o mesmo exemplo de anteriormente, o output depois da função “tr” seria “enp7s0 lo”.

Após obtermos os nomes das interfaces vamos buscar os dados individualmente através dos ficheiros presentes na pasta /sys/class/net/nome_da_interface, sendo que nesta pasta podemos encontrar para cada interface os valores atuais de "rx_bytes" (/sys/class/net/nome_da_interface_rx_bytes) e tx_bytes (/sys/class/net/nome_da_interface_tx_bytes). Este processo é feito com o auxílio da função interna à bash, "cat", guardando o seu output nas variáveis "tx" e "rx" de cada interface de rede.

Sabendo então como obter os valores, cada lista que armazena os dados relativos às interfaces têm o seguinte conteúdo:

1) "rx_bytes_i" – bytes recebidos antes da chamada da função sleep x, sendo "x" os segundos passados como argumento;
2) "tx_bytes_i" – bytes enviados antes da chamada da função sleep x, sendo "x" os segundos passados como argumento;
3) "rx_bytes_f" – bytes recebidos depois da chamada da função sleep x, sendo "x" os segundos passados como argumento;
4) "tx_bytes_f" – bytes recebidos depois da chamada da função sleep x, sendo "x" os segundos passados como argumento;

As outras listas "rx" e "tx" armazenam as diferenças e divisões, como já previamente descrito na secção "I) Estruturas de dados utilizadas e a lógica de organização de dados".

V. Conversão de bytes para kilobytes e megabytes

Para a conversão de bytes para kilobytes e megabytes foram usadas estruturas "if-then-elif-then" para verificar se as flags de conversão se encontravam ativas.

Para atestar à divisão com vírgula flutuante foi utilizada a "bc" ("basic calculator") da bash em cada conversão, utilizando a função "echo" para armazenar o resultado da avaliação passada entre aspas à "bc" em variáveis.

Caso a flag "-b" estivesse ativa não há alteração dos valores que se encontram nas listas "tx_bytes", "rx_bytes", "total_tx_bytes" e "total_rx_bytes".

Caso a flag "-k" estivesse ativa os valores que se encontram nas listas "tx_bytes", "rx_bytes", "total_tx_bytes" e "total_rx_bytes" são divididos com o auxílio da "bc" por mil.

Caso a flag "-m" estivesse ativa os valores que se encontram nas listas "tx_bytes", "rx_bytes", "total_tx_bytes" e "total_rx_bytes" são divididos com o auxílio da "bc" por um milhão.

Os valores das listas "total_rx_bytes" e "total_tx_bytes" só são incrementados com o valor de "rx_bytes" e "tx_bytes" já convertidos, tendo em consideração a "flag" de conversão ativa. Se não fizermos esta distinção de incrementação, caso a flag "-l"

esteja ativa os valores das listas totais irão tender para 0 já que a conversão de byte para megabyte, por exemplo, seria aplicada em cada iteração.

VI. Algoritmo de ordenação dos dados (Bubble Sort)

Para a ordenação dos dados foram utilizadas estruturas “if-then-elif-then” para averiguar se as flags de ordenação se encontravam ativas.

Caso a flag “-t” estivesse ativa os valores da lista “tx_bytes” serão ordenados por ordem decrescente através do algoritmo de ordenação “Bubble Sort”.

Caso a flag “-r” estivesse ativa os valores da lista “rx_bytes” serão ordenados por ordem decrescente através do algoritmo de ordenação “Bubble Sort”.

Caso a flag “-T” estivesse ativa os valores da lista “trate” serão ordenados por ordem decrescente através do algoritmo de ordenação “Bubble Sort”.

Caso a flag “-R” estivesse ativa os valores da lista “rrate” serão ordenados por ordem decrescente através do algoritmo de ordenação “Bubble Sort”.

Esta ordenação é feita não nas listas “tx_bytes”, “rx_bytes”, “trate” e “rrate”, mas sim na lista “ordem”. Como afirmado anteriormente, a ordem das primeiras quatro listas mencionadas é definitiva e assegurada durante a execução do script. O que, de facto, é ordenado no trecho de código de ordenação são os valores dos índices da lista “interfaces” presentes na lista “ordem”. Logo, apesar de serem avaliados os valores das listas “tx_bytes”, “rx_bytes”, “trate”, “rrate” o que é alterado são os valores da lista ordem.

Apesar do “Bubble Sort” não ser o melhor algoritmo de ordenação devido à sua elevada “time complexity”, sendo nos piores casos, O(n^2), reconhecemos que devido ao caráter dos dados, sendo eles um número reduzido, não justifica aplicarmos um algoritmo de ordenação mais refinado para existir diferenças não percetíveis ao “olho-humano”. Logo, a simplicidade do algoritmo pareceu-nos adequado ao problema ao qual nos depáramos.

Por último, caso a flag “-v” estiver ativa e a variável “did_order_change” (variável que toma o valor de 1 quando houve alteração na lista “ordem”) for igual a 1 a lista “temp_ordem” vai consistir no conteúdo da lista “ordem”, mas na ordem inversa. Este efeito foi obtido a partir da manipulação de índice. De seguida, o conteúdo que se encontra em “temp_ordem” é passado para “ordem”.

VII. Impressão dos dados no terminal

A impressão dos dados no terminal foi feita utilizando a função “printf” que permite a formatação do conteúdo de forma organizada e consistente, utilizando os caracteres especiais “%” que permitem caracteres adicionais de alinhamento e de reserva de “espaços” para os conteúdos passados como argumentos da função.

É neste passo que fazemos a avaliação do que é realmente impresso, apesar de termos tudo calculado internamente, só mostramos o que é pedido em termos das flags “-c”, “-p” e “-l”.

Neste passo percorremos um “for-loop” n_interfaces_display vezes, que caso “-p” estiver inativo, terá o valor padrão do número total de interfaces. Dentro do “loop” vamos obter o valor para a variável “ord” a partir de “ordem[i]”. O valor de “ord” vai ser utilizado para obter os valores da interface de rede que se encontra no índice “ord” na lista “interfaces”. Caso a flag “-c” esteja ativa, o nome da interface de rede, interfaces[ord], vai ser avaliada com a expressão regular armazenada na variável “regex”, se a avaliação retornar o valor 1, então a interface de rede e os valores associados são impressos, caso contrário, não são impressos e o loop vai para a próxima iteração ou termina.

Caso a flag “-l” estiver ativa então o processo anterior corre normalmente, no entanto a impressão do cabeçalho é feita só uma vez com o auxílio da variável, já anteriormente mencionada, “count”, usada para contar o número de iteração do “while-loop”. Para além disso, também são adicionadas duas colunas, “TXTOT” e “RXTOT” que contém os valores totais acumulados em cada iteração.

VIII. O código relacionado com a flag “-l”

Devido à flag “-l” tivemos que utilizar a estrutura de repetição “while-loop” que repete as chamadas ao sistema para a recolha dos dados de cada interface, tx e rx bytes, indefinidamente, caso a flag esteja ativa. Se a flag não estiver ativa, na primeira iteração do loop, no código que imprime os dados, há um “break” que termina o loop.

Para além do “while-loop” também foi necessário implementar uma variável que contabiliza o número de iterações do loop para ser possível imprimir somente uma vez o cabeçalho.

IX. Erros e testes

No decorrer do trabalho deparámos-mos com diversos erros ao fazer a testagem o que nos levou a reforçar o controlo dos argumentos passados e utilizar estruturas condicionais para resolver “bugs”.

Na execução do nosso script, é possível ser confrontado com as seguintes mensagens de erro que levam à terminação do programa com “exit code” igual a 1:

- “Error: Invalid number of arguments. (Min: 1; Max: 9)”
- “Error: time argument must not be equal to 0.”
- “Error: When passing only 1 argument make sure that it is an integer.”

- “Error: Two or more conversion flags were used.”
- “Error: Number of interfaces must be lower or equal than X.”, sendo X o número de interfaces de rede do computador;
- “Error: Number of interfaces must be an integer.”;
- “Error: Two or more sorting flags were used.”;
- “Error: Invalid flag was passed.”;
- “Error: Same flag passed more than once.”;
- “Error: Arguments passed are creating conflict.”;
- “Error: If not succeeding ‘-c’, characters are not permitted as arguments.”;
- “Error: flag argument is creating conflict with rate time.”;
- “Error: Invalid number of integers according to active flags.”;
- “Error: time argument must be an integer.”

A testagem consistiu maioritariamente da passagem de argumentos de todos os tipos e combinações. Por exemplo, passámos todos os argumentos possíveis, várias combinações de flags de ordenação e conversão, juntamente com a flag “-l” para observar se o ciclo de repetição não interferia com os algoritmos de ordenação/conversão, passando também argumentos que não se encontravam tabelados e argumentos incorretamente posicionados.

Bugs com os quais nos deparámos durante a execução do trabalho:

1) Aquando da ativação da flag “-l” o cabeçalho imprimia-se múltiplas vezes

Solução: resolvemos este problema ao utilizar um contador de interações do “while-loop”. Na primeira iteração o cabeçalho era impresso, no entanto, na segunda e nas restantes já não era.

2) Aquando da ativação da flag “-l” juntamente com o “-r”, o “reverse” era aplicado em cada iteração;

Solução: resolvemos este problema ao utilizar o mesmo contador do erro 1), na primeira iteração do “while-loop” era aplicado o filtro de “reverse”, mas nas restantes o filtro não era corrido.

3) Aquando da ativação das flag de conversão/ordenação tivemos problemas devido aos cálculos que implicavam cálculos com vírgula flutuante;

Solução: resolvemos este problema ao utilizar a “basic calculator” da bash que permite operações com vírgula flutuante. Nos casos em que era esperada a adição, subtração ou divisão de valores com vírgula flutuante foi utilizada a “bc”, mas nos casos em que era assegurado que os valores eram inteiros, utilizamos a aritmética inteira interna à bash.

4) Ao passar múltiplos argumentos de conversão/ordenação o programa escolhia sempre por “default” o “-b” para a conversão e o “-t” para a ordenação;

Solução: no trecho de código do “getopts” implementámos diversas estruturas condicionais que avaliavam quais as flags que estavam ativas, imprimindo uma mensagem de erro e terminando a execução do programa quando se encontravam flags que coincidiam.

5) O não controlo do conteúdo associado à flag “-p” fazia com que o nosso output não fosse correto, pois a primeira interface imprimida seria repetida a diferença entre o número passado como “n_interfaces_display” e “#interfaces[@]" vezes

Solução: resolvemos este problema ao controlar o argumento passado após a flag “-p”, limitando o número inferiormente e superiormente, não podendo ser menor do que 2 (nome do script + o argumento dos segundos, argumento obrigatório como indicado no guião) e não podendo exceder o valor de 10 que correspondia ao máximo de argumentos válidos passados (os 2 argumentos já mencionados + 1 argumento de conversão + 1 argumento de ordenação + 1 argumento do “reverse” + 1 argumento do loop + 2 argumentos relacionados com a regex + 2 argumentos relacionados com a flag “-p”).

No decorrer do desenvolvimento do trabalho foram realizados inúmeros testes para ir comprovando a eficácia do código desenvolvido.
Todos os testes exemplo que estão no guião do trabalho foram realizados. Para além dos apresentados no guião foram realizados testes para comprovar que o controlo de erros desenvolvido em código estava a funcionar corretamente e todas as flags e combinações que corriam o programa com sucesso, experimentando o bom funcionamento

Alguns testes extra que efetuámos:
./netifstat.sh -t -r 10 (diversas flags de sort=error)
./netifstat.sh -k -m 10 (diversas flags de conversão=error)
./netifstat.sh -c “l.*” -p 2 -k -t -v -l 10 10 (número inválido de argumentos=error)
./netifstat.sh h (quando é passado apenas um argumento este tem de ser inteiro)
./netifstat.sh 0 (quando é passado apenas um argumento este tem de ser diferente de 0)
./netifstat.sh -t -h 10 (flag inválida)
./netifstat.sh -T -v -l 10
./netifstat.sh -T -v -l -k 10
./netifstat.sh -k -t -c "l" -p 2 -v -l 10
./netifstat.sh -t 0 (tempo tem de ser maior que 0)

X. Conclusão

Com este trabalho tivemos que nos documentar acerca do funcionamento de diversas funções internas à bash, sobre a sintaxe da programação em shell, sobre algoritmos de ordenação e da organização dos ficheiros num sistema operativo, o que nos permitiu, por sua vez, aprender e consolidar conhecimentos importantes. Para além disso, a realização de testes e de controlo de código, tentando assegurar de melhor forma a consistência do programa, fez-nos aperceber que realmente, em programas dependentes de argumentos passados na linha de comandos, é necessário haver um cuidado acrescido relativamente ao processamento e validação dos argumentos, o que não acontece tanto em cenários em que é utilizada uma interface gráfica com botões, onde o "input" é feito através de cliques.

XI. Referências

- https://www.baeldung.com/linux/use-command-line-arguments-in-bash-script (arguments in bash)
- https://www.geeksforgeeks.org/bc-command-linux-examples/ (bc command)
- https://linuxhint.com/bash_tr_command/ (tr command)
- https://stackoverflow.com/questions/31221343/override-invalid-option-message-with-getopts-in-bash (default case in getopts)
- https://tldp.org/LDP/abs/html/comparison-ops.html (comparion operators)
- https://www.geeksforgeeks.org/cut-command-linux-examples/ (cut command)
- https://www.geeksforgeeks.org/bubble-sort/ (Bubble Sort)
- https://stackoverflow.com/questions/16489809/emulating-a-do-while-loop-in-bash (do while emulation)
- https://stackoverflow.com/questions/21112707/check-if-a-string-matches-a-regex-in-bash-script (regex matching)
- https://stackoverflow.com/questions/16483119/an-example-of-how-to-use-getopts-in-bash (getopts)
- https://linuxize.com/post/bash-printf-command/ (printf)
- https://stackoverflow.com/questions/8654051/how-can-i-compare-two-floating-point-numbers-in-bash (floating point arithmetic)
- https://unix.stackexchange.com/questions/605698/bash-use-flags-without-inserting-arguments (flags without argument)
- https://stackoverflow.com/questions/7529856/retrieving-multiple-arguments-for-a-single-option-using-getopts-in-bash (compreending getopts command)
- https://stackoverflow.com/questions/6450152/getopt-value-stays-null (compreending getops command)
- https://unix.stackexchange.com/questions/151654/checking-if-an-input-number-is-an-integer (regex matching integer)

Para além das consultas online relativamente a comandos e instruções internas à bash foi usado o manual da bash (man ____) como auxílio durante a execução do trabalho.