

**Sistemas Operativos**

**Estatística de Processamento em Bash**

**Professor:**

Nuno Lau ([nunolau@ua.pt](mailto:nunolau@ua.pt))

**Pedro** **Sobral**, 98491, P3

**Daniel Figueiredo**, 98498, P5

**As distribuições de percentagem são iguais para os membros do grupo**

07/12/2020

**Índice**

[1 - Introdução 3](#_Toc57849585)

[2 - Estrutura do Código 4](#_Toc57849586)

[2.1 – Declaração de variáveis globais 4](#_Toc57849587)

[2.2 – Tratamento de opções 5](#_Toc57849588)

[2.3 – Listar os Processos 6](#_Toc57849589)

[2.3.1 – Listar os Processos STATUS 7](#_Toc57849590)

[2.3.2 – Listar os Processos IO 8](#_Toc57849591)

[3 – Seleção de Processos 9](#_Toc57849592)

[3.1 – Seleção de Processos – Opção “-u“ 9](#_Toc57849593)

[3.2 – Seleção de Processos – Opção “-c” 9](#_Toc57849594)

[3.3 – Seleção de Processos – Opção “-s” 9](#_Toc57849595)

[3.4 – Seleção de Processos – Opção “-e” 9](#_Toc57849596)

[Conclusão 10](#_Toc57849597)

[Bibliografia 11](#_Toc57849598)

# **1 - Introdução**

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

A implementação do código bash será feita através do IDE *VSCode*, pois é um editor com que ambos os membros grupo já estão familiarizados derivados a unidades curriculares e projetos passados. A execução de todo o trabalho foi suportada através de um [repositório no GitHub](https://github.com/TheScorpoi/SO_Trabalho01)1, o que facilita em muito todo o *workflow* da realização do mesmo.

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

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

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

# **2 - Estrutura do Código**

## 2.1 – Declaração de variáveis globais

Primeiramente, para o tratamento da informação, pensamos utilizar *arrays* associativos, pois são estruturas de dados que permitem referenciar vários dados por um único nome, deste modo, para nos referirmos a um determinado valor utilizamos uma “chave”, que está diretamente ligada a um valor. Assim achamos por bem, criar este tipo de estruturas de dados e inicializamos os *arrays arrayAss* e *argOpt* (Fig.1)*.*

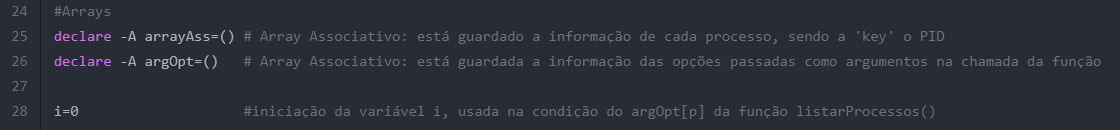


Figura - Inicialização de variáveis globais.

* ***arrayAss*** – *array* associativo que contém informações acerca de todos os processos que, posteriormente tenham passado pelos processos de validação no nosso código, tem como *key* o *PID* de cada processo, não havendo, assim, colisão de informação
* ***argOpt*** - *array* associativo onde estão guardadas as informações das opções passadas como argumento, quando se corre o programa, sendo a *key,* a opção passada pelo utilizador e o seu *value* o argumento da mesma.

## 2.2 – Tratamento de opções

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

Uma imagem com texto

Descrição gerada automaticamente

Figura 2 - Função opcoes()

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

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

## 2.3 – Listar os Processos

A função *listarProcessos* (Fig.3)é uma das principais funções do *script*, sendo que é aqui que se lê as informações dos processos e se trata desses dados de forma primordial.



Figura - Função ListarProcessos()

Começamos por *printar* o cabeçalho da nossa tabela, visto que este será sempre impresso, e só queremos que este seja *printado* uma vez a cada execução do *script.* Seguidamente iniciamos um ciclo *for*, que nos irá permitir percorrer todos os processos que estão ativos no nosso computador, cada processo fica na variável *entry* a cada iteração do ciclo *for*, este ciclo itera sobre o diretoria */proc/[[:digit:]]\*,* onde esta última expressão nos irá dar todos os processos em que as diretorias são dígitos.

Ir ao terminal e correr o cd /proc ou la como é, e mostrar as diretorias so com os números, por foto disso

Agora seguem-se algumas condições para o tratamento dos dados, várias expressões condicionais (Fig.4). O primeiro if [[ -r $entry/status && -r $entry/io ]], serve para, caso não tenhamos permissões de acesso a uma determinada informação do processo, não o vamos listar. Segue-se mais um if [[ -f $entry/comm ]], neste if é decidido logicamente se o processo possui um comm (O QUE É QUE O -F FAZ??).



Figura 4 - if's de permissões

### 2.3.1 – Listar os Processos - STATUS

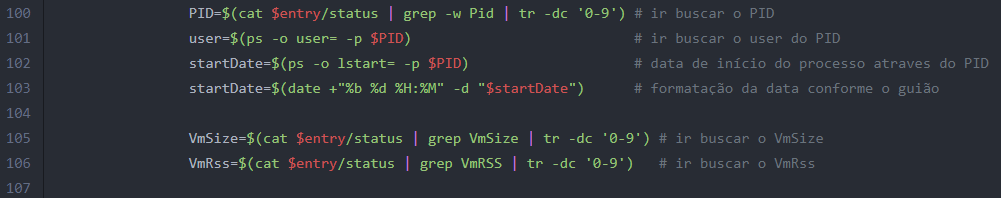
Agora chegamos a uma fase do código em que iremos guarda em variáveis locais, alguns atributos necessários para a realização da tabela proposta no guião. Começamos pelo PID, o PID é o identificador do processo em questão, sendo este um valor muito importante pois será usado no procedimento de criação de outra variáveis, p.e. user, startDate.

Figura – Atributos vindos do STATUS

Usamos o cat /$entry/status, pois é nesse *path* que o PID, se encontra, usamos um *grep -w Pid*, para que devolve-se só a linha que possuía o PID, e um *tr -dc ‘0-9’*, para devolver somente a parte numérica do PID.

Para atribuirmos à variável *user*, o utilizador do processo, fizemos o comando ps, e com o PID, obtivemos o utilizador. Para a data de início do processo, o mecanismo foi o mesmo, com o PID, e o comando ps, guardados a data inicial do processo na variável *startDate*. Quanto à variável *startDate*, a seguir fizemos uma formatação da mesma, para que fique no formato: mês, dia, horas, minutos.

Em seguida fomos buscar o VmSize, e o VmRss, que correspondem à memoria FISICA E VIRTUAL??????\*\*\*\*\*, respetivamente, o processo de filtragem da informação foi semelhante ao do PID, usamos um *grep* e um *tr* para que só fossem devolvidos os valores dessas 2 variáveis.

### 2.3.2 – Listar os Processos - IO

Aqui continuamos a guardar em mais variáveis valores necessários para completar a nossa tabela (Fig.6).

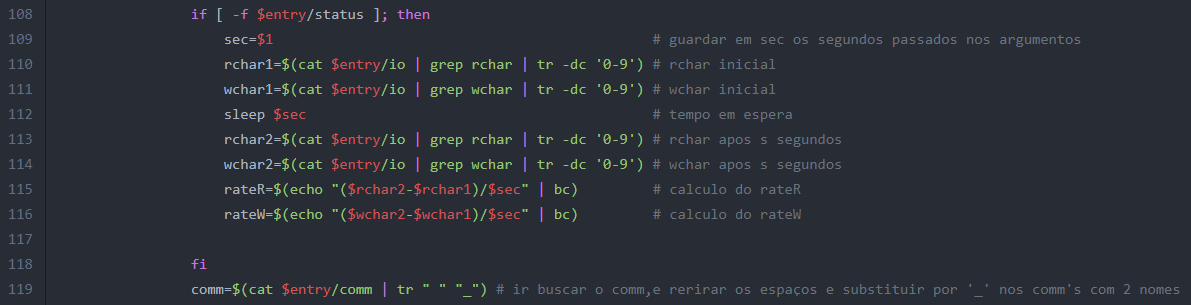
Fazemos o if [[ -f $entry/status ]], que de uma forma genérica verifica de o *file* *$entry/io* existe.

Figura - Atributos vindos do IO

Guardamos o argumento passado á função *listarProcessos*, $1, na variável sec, que são os segundos em que o *script* irá executar um sleep. Em seguida vamos filtrar a informação relativamente ao *rchar* e ao *wchar*, que são os bytes que o processo leu (*rchar1*) e escreveu (wchar1), respetivamente. Fazemos um sleep de sec segundos, e voltamos a ler os valores de *rchar* e *wchar, e guardamos nas variáveis rchar2 e wchar2*. De seguida, vamos calcular o *rateR, e o rateW*, que são a taxa de leitura e escrita num determinado intervalo de tempo, neste caso, no intervalo *sec*. Em ambos o cálculo é simples, uma subtração do último valor lido com o valor lido primeiro, a dividir pelo número de segundos em que o *sleep* esteve ativo. Este cálculo é feito tanto para o rateR como para o rateW.

Em seguida criamos a variável *comm*, onde irá ficar guardado o nome do processo em analise, fizemos a substituição do espaço por um ‘\_’, pois durante a realização do trabalho prático havia alguns processos em que o *comm*, eram duas palavras, sendo que depois na hora de ordenar, a tabela não ordenava corretamente, derivado a isso, desta forma o problema ficou instantaneamente resolvido.

# **3 – Seleção de Processos**

Agora chegamos a uma fase onde é capital tratar da informação de acordo com os argumentos passados no terminal, ou seja, só vamos adicionar um determinado processo e toda a sua informação ao array, se se verificarem determinadas condições passadas no terminal (Fig.7). De forma mais informal, só vamos adicionar informação ao array, se de acordo com as opções passadas no terminal, se verificarem nos if’s. (acho q n faz sentido a frase, ver dps)

POR MAIS PALHA AQUI SFF, obg e volte sempre

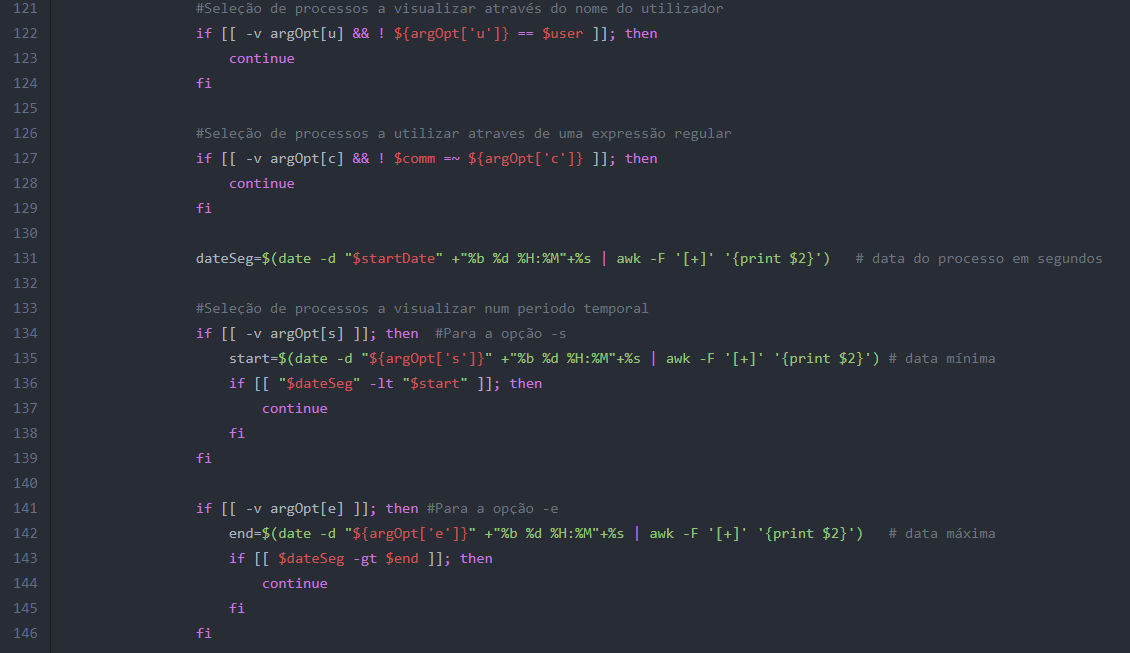


Figura - Visão geral do código para a seleção de processos

O raciocínio aplicado para a seleção dos processos, foi semelhante para todas as opções verificadas, através de uma expressão condicional vamos verificar se a opção se encontra no array *argOpt,* que é o array onde ficam guardadas as opções e os argumentos caso os tenham, quando passados no terminal. Se a opção passada se verificar, e se o argumento dessa opção for de acordo com o que está a ser verificado para essa opção, então esse processo é adicionado ao array *ArrayAss.*

## 3.1 – Seleção de Processos – Opção “-u“

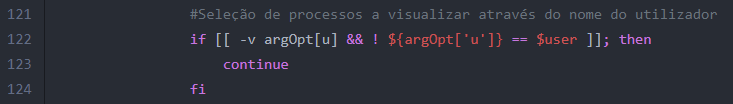
Para a opção “-u” 

Figura - Opção "-u"

## 3.2 – Seleção de Processos – Opção “-c”

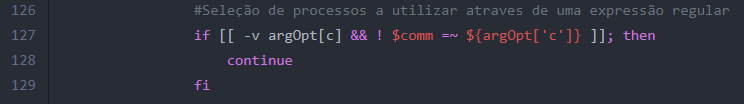


Figura - Opção "-c"

## 3.3 – Seleção de Processos – Opção “-s”

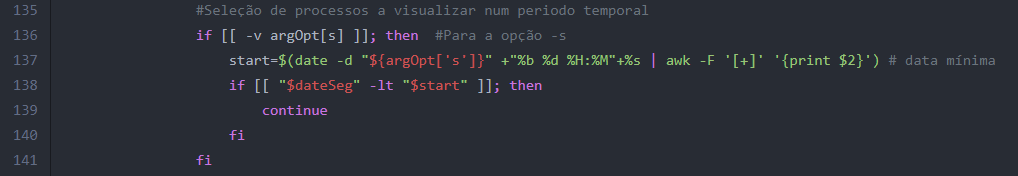


Figura - Opção "-s"

## 3.4 – Seleção de Processos – Opção “-e”

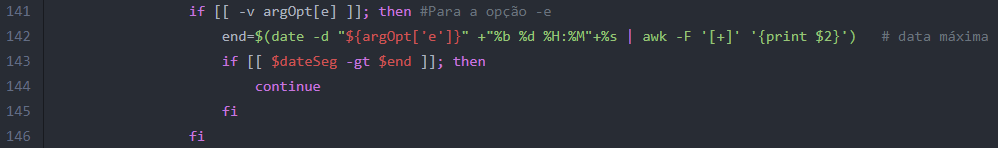


Figura - Opção "-e"

# Conclusão

Ao longo da realização deste trabalho

# Bibliografia

- Para a realização deste trabalho prático também consultamos os slides disponibilizados pelo Professor na página do *e-learning* da unidade curricular Sistemas Operativos.

<https://man.cx/bash>

<https://stackoverflow.com/>

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

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

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