Sistemas Operativos

Ano Letivo: 2020/2021

Tomé Lopes Carvalho - 97939

Gonçalo Fernandes Machado - 98359

Trabalho 1

Estatísticas de processos em bash

Descrição geral da abordagem:

1. Verificar se existem os ficheiros “data.txt”, “data\_sorted.txt” e “tmp.txt”, e caso existam, proceder à sua remoção.
2. Verificar os argumentos e opções introduzidas pelo utilizador. Criar variáveis que guardam informação sobre as opções. Analisar as opções introduzidas e alterar as variáveis consoante as mesmas. Verificar se o argumento obrigatório foi introduzido, se foram introduzidas opções incompatíveis e se os argumentos das opções, se existirem, são válidos.
3. Percorrer todos os diretórios em /proc. Adicionar à lista de PIDs todos os diretórios cujo nome seja um número (correspondente ao ID do processo).
4. Para cada PID, recolher a seguinte informação: VmSize (MEM), VmRSS (RSS), user (USER), rchar (READB), wchar (WRITEB), DATE. Remover os processos que não têm */proc/pid/status* e/ou */proc/pid/io* readables, em que *pid* é o PID do processo, ou que não têm atributos VmSize/VmRSS. Verificar também, de acordo com as opções de filtragem (passo 2), se o processo será apresentado ou não.
5. Aguardar *s* (valor introduzido pelo utilizador - último argumento) segundos, voltar a recolher VmSize e VmRSS dos processos, de modo a poder calcular RATER e RATEW.
6. Guardar a informação que vai ser apresentada num ficheiro (*data.txt*).
7. Ordenar os processos de acordo com as opções introduzidas pelo utilizador (utilizando algumas das variáveis do passo 2).
8. Imprimir a informação no terminal.
9. Remover o ficheiro em que está guardada a informação apresentada.

Funcionamento de cada passo:

1.

É criado o array *files*, que consiste numa lista de strings que correspondem aos nomes dos ficheiros que irão ser utilizados no script. É executado um ciclo *for* onde se verifica se algum dos ficheiros existe, e se existir procede-se à sua remoção. Isto é feito porque se interrompermos o script enquanto está a correr ficamos com um dos ficheiros criados, e ao correr novamente o processo sem esta verificação, seria adicionado o conteúdo do novo processo ao ficheiro com o conteúdo do antigo, imprimindo no final informação criada na execução anterior.

2.

É verificado se o último argumento (${@: -1}), o argumento obrigatório, é um número, utilizando a regular expression *valid\_number\_regex*, sendo que caso não seja, é apresentado o guia de utilização e o programa termina.

Se forem usadas as opções -s e/ou -e, verifica-se, através do comando “date -d”, se as datas introduzidas como argumentos são válidas.

Caso tenha sido introduzida a opção -p, verifica-se se o seu argumento é um número inteiro não-negativo.

São criadas as variáveis *usr\_filter*, *comm\_filter*, *min\_date\_filter*, *max\_date\_filter*, *sort\_col*, *reverse\_sort*,  *max\_amount\_filter* e *n\_sort\_opts\_used*. No caso das 4 primeiras e da penúltima, serão colocados os argumentos das opções -u, -c, -s, -e, -p caso forem utilizadas (no caso de *min\_date\_filter* e *max\_date\_filter*, as datas são convertidas para o formato epoch). O valor de *sort\_col* será alterado para a coluna de informação correspondente. *reverse\_sort* será igual a 1 caso seja utilizada a opção -r, caso contrário será igual a 0.

O valor de *n\_sort\_opts\_used* é incrementado por 1 cada vez que se verifica que foi utilizada uma opção de *sort* (à exceção da opção de reverse). Sendo estas incompatíveis, se o valor desta variável for maior que 1 o programa termina e apresenta o guia de utilização.

3.

Inicialmente, é criado um array vazio, *PIDS*.

Utilizando um ciclo *for*, são percorridos todos os diretórios de /proc. Guarda-se o nome do diretório na variável *dir\_name*. Averigua-se, através da mesma *regex* do passo 1, se o nome do diretório é um número. Caso seja, o nome, que corresponde ao PID do processo, é adicionado ao array *PIDS*.

4.

São criados os arrays *users*, *comm*, *vmSize*, *vmRss*, *rI*, *wI*, *rF*, *wF*, *rC*, *dates*, *dates\_epoch*, cujos índices corresponderão aos *PID* dos processos.

É criado um array *PIDS\_2*, onde serão colocados os *PID* dos processos selecionados no ciclo *for* que se segue.

No ciclo *for*, é feito o seguinte para cada processo (*i* é o PID do processo):

1. Se não existir ficheiro *status* e/ou ficheiro *io* ou se pelo menos um destes ficheiros não for readable (verificado principalmente através de [ -r ]), não será adicionado a *PIDS\_2*. Como alguns processos podem ser considerados readable pelo [ -r ] mas negar a permissão (por exemplo, *gnome-shell*) quando se tenta ler, é executada uma segunda verificação, desta vez correndo a função check\_readable\_pid() , que corre o comando cat no ficheiro com output redirecionado para o diretório dev/null e verifica se o comando correu sem erros.
2. Utilizando os comandos *cat*, *grep*, *cut*, *xargs* (remove whitespaces), definimos os atributos *vmSize* e *vmRSS* de cada processo. Se um destes ou ambos forem nulos, não será adicionado o processo a *PIDS\_2*.
3. Utilizando os comandos *ps*, *awk* e *comm*, definimos os atributos *USER* e *COMM* de cada processo. Aplica-se após a definição de cada um os filtros de nome de utilizador
4. Utilizando o comando *ls -ld*, obtemos a data e fazemos pipe para um *awk* que a coloca num formato “mês;dia;hora:minutos” (por exemplo, “Nov;28;18:16”). Guardamos a data neste formado no array *dates* e a mesma convertida em formato epoch (através do comando «LC\_ALL=en\_US.utf8 date -r $inputdate “+%s”», onde “LC\_ALL=en\_US.utf8” garante que a data que o comando retorna vem no formato de inglês americano) no array *dates\_epoch* (para facilitar a comparação de datas). Aplicam-se os filtros de de data mínima e data máxima
5. De modo semelhante ao utilizado para obter *vmSize*[*$i*] e *vmRSS*[*$i*], obtemos *rI[$i]* e *wI[$i]*, os bytes de leitura e escrita, respectivamente.
6. Colocámos antes do ciclo *for* uma variável *max\_comm\_length*, que é alterada cada vez que se encontra um processo com *COMM* mais comprido que o seu valor. Serve para lidar com o espaço necessário para imprimir na terminal todos os *COMM* sem desformatar o restante texto (nota sobre este passo nos testes)

5.

Utiliza-se o comando *sleep*, com argumento igual ao último argumento introduzido ao iniciar o script.

É feito outro ciclo *for*, iterando sobre *PIDS\_2*, onde, em primeiro lugar, é verificado novamente se o ficheiro *io* de cada processo é readable com [ -r ] e depois é utilizada a função *check\_readable\_pid\_io*, pelos mesmos motivos mencionados no ponto 4.1.

São lidos de novo os valores de *READB* e *WRITEB* para cada processo, com o fim de calcular as taxas de leitura e de escrita (*RATER* e *RATEW*).

É calculada a diferença entre *READB/WRITEB* finais (*rF[$i]*/*wF[$i]*) e *READB/WRITEB* iniciais (*rC[$i]*/*wC[$i]*).

Se o número de segundos a aguardar introduzido não for 0, as rates são calculadas através da divisão das respectivas diferenças pelo valor do argumento introduzido. Utilizamos, nesta situação, o comando *bc* para realizar divisões com vírgula flutuante.

Se o número de segundos a aguardar introduzido for igual a 0, não podemos realizar o cálculo. Decidimos apresentar o número de bytes lidos/escritos durante a execução do código.

6.

Ainda no ciclo *for* anterior, é guardada no ficheiro *data.txt* toda a informação que diz respeito ao processo sobre o qual se está a iterar. As linhas deste ficheiro são ordenadas alfabeticamente (por *COMM*, uma vez que é o primeiro atributo dos processos) para *data\_sorted.txt*. É removido *data.txt*.

7.

São analisadas as variáveis que dizem respeito à ordenação, começando pelas que se tratam de ordenação numérica com base numa coluna correspondente a um atributo (a coluna é indicada pela variável *sort\_col*, se esta estiver vazia ( [ -z “$*sort\_col*” ]) manter-se-á a ordenação por ordem alfabética de *COMM*).

De seguida, se a variável *reverse\_sort* tiver sido alterada para 1 (opção -r), inverte-se a ordem das linhas do ficheiro *data\_sorted.txt*.

Se tiver sido utilizada a opção -p, utiliza-se o comando *head* com o argumento dessa opção para retirar todas as linhas exceto as *n* primeiras, *n* sendo o argumento de -p.

Todos os métodos de sort mencionados acima começam primeiro por colocar a informação num ficheiro *tmp.txt*, e movendo o conteúdo do *tmp.txt* para o *data\_sorted.txt*, ou seja, dando overwrite ao *data\_sorted.txt* e removendo o *tmp.txt*.

8.

Finalmente, utiliza-se *printf* para imprimir o header e a informação de cada processo, através de um ciclo *while* que percorre as linhas de *data\_sorted.txt*.

9.

O ficheiro *data\_sorted.txt* é removido no final do programa.

Testes

Os testes foram realizados em:

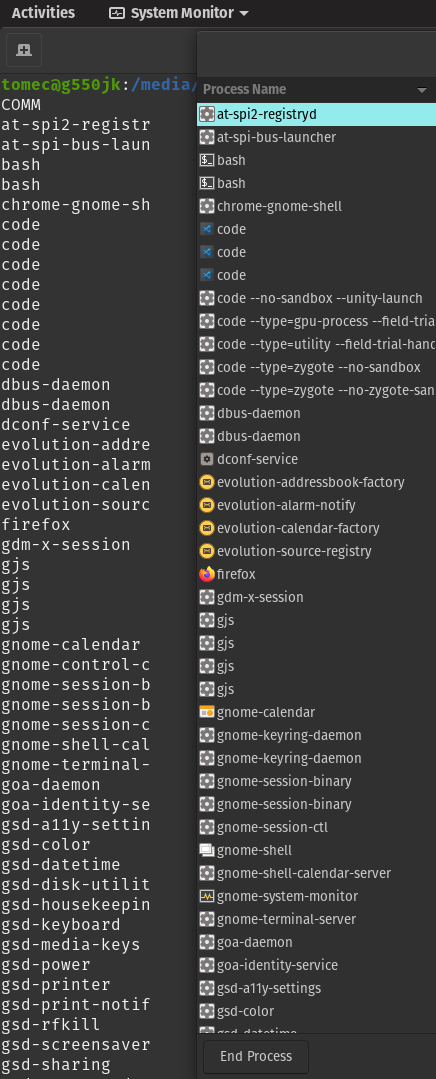
OS: Pop!\_OS 20.04 LTS x86\_64 (baseado em Ubuntu 20.04)

Kernel: 5.8.0-7630-generic

Shell: bash 5.0.17

DE: Gnome

Terminal: gnome-terminal



Comparação dos processos apresentados com o System Monitor do Gnome