Trabajo de Diseño y Administración de Sistemas Operativos

Alumno: Nombre Apellido1 Apellido2

DNI: 00000000-X

Centro Asociado: XXXXX

Teléfono de contacto: 000000000

Email: xxx@uned.es

# Primera PED

## Introducción

Una forma común de obtener información sobre los diversos procesos y estadísticas de uso en sistemas UNIX es utilizar el comando top, una suerte de administrador de tareas que muestra dicha información de forma organizada y fácil de leer.

El objetivo de esta práctica será implementar un script en Bourne Again Shell que emule la funcionalidad básica del comando TOP anteriormente descrito, auxiliándose para tal tarea de los directorios **proc/PID** presentes en el sistema de archivos UNIX, para obtener la información necesaria para realizar tanto identificación de procesos como cálculos de porcentajes de uso.

A continuación, se muestra la tabla de referencia de donde se obtuvieron los datos requeridos por la práctica.

|  |  |  |  |
| --- | --- | --- | --- |
| Dato | Ubicación | | Detalles |
| Información de cada PID | | | |
| PID | /proc/PID/stat[0] | | Para este y todos los datos que hacen referencia a una posición del archivo **stat** se hace la aclaración de que dicha notación corresponde a que la información del archivo stat fue recuperada mediante readarray, por lo que cada campo paso a ser una posición de arreglo. |
| USER | Obtenido a partir del comando getent utilizando como user id el localizado en /proc/pid/Status | | Se utilizó el comando getent a través de passwd y el user id del archivo status para obtener el usuario. |
| Prioridad | /proc/PID/stat[17] | |  |
| Memoria virtual | /proc/PID/stat[22] | |  |
| Estado | /proc/PID/stat[2] | |  |
| %CPU |  | | Se calculó mediante diferencias del valor Tiempo antes y después de sleep 1, este valor se ajustó mediante el valor CLK\_TCK que para este caso era 100 y así se obtuvo un porcentaje. |
| %Memoria |  | | Para cada proceso se calculó el porcentaje de memoria del mismo mediante stat[22] y la memoria total de meminfo. |
| Tiempo | /proc/PID/stat[14] + /proc/PID/stat[15] | | Se obtuvo el dato de tiempo en modo usuario y en modo kernel y se sumaron para el tiempo total. |
| Comando | /proc/PID/stat[1] | |  |
| Información de cabecera | | | |
| Memoria total | | **/proc/meminfo[0]** |  |
| Memoria libre | | **/proc/meminfo[1]** |  |
| Memoria utilizada | | **/proc/meminfo[0] - /proc/meminfo[0]** | Se restaron los valores de memoria total y libre para obtener la memoria en uso. |
| %CPU | |  | Se obtuvo mediante la suma de porcentaje de CPU de cada PID. |

La documentación pertinente de los métodos utilizados se muestra a continuación:

Readarray

readarray - Read lines from the standard input into the indexed array variable array, or from file descriptor fd if the -u option is supplied

Read lines from the standard input into the indexed array variable array, or from file descriptor fd if the -u option is supplied.  The variable MAPFILE is the default array. [1]

Grep

**grep** searches for PATTERN in each FILE. A FILE of "**-**" stands for standard input. If no FILE is given, recursive searches examine the working directory, and nonrecursive searches read standard input. By default, **grep** prints the matching lines.

In addition, the variant programs **egrep**, **fgrep** and **rgrep** are the same as **grep -E**,**grep -F**, and **grep -r**, respectively. These variants are deprecated, but are provided for backward compatibility. [2].

Bc

 Is a language that supports arbitrary precision numbers with interactive execution of statements. bc starts by processing code from all the files listed on the command line in the order listed. After all files have been processed, bc reads from the standard input. All code is executed as it is read. (If a file contains a command to halt the processor, bc will never read from the standard input.)

The most common use of bc is within a shell script, using a "here" document to pass the program details to bc. [3]

Sort

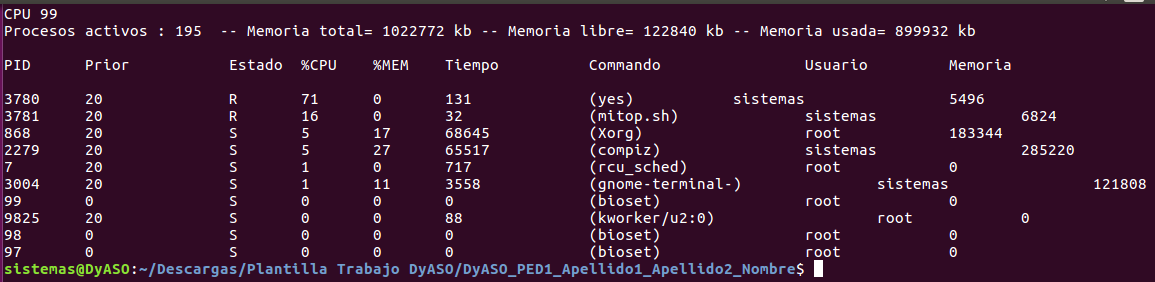
SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically. [4]

## Implementación

|  |
| --- |
| mitop.sh |
| *#!/bin/bash*  *#auxiliar functions section*  *function printHeaderData {*  *#All the mem information is located in the file meminfo*  *readarray memData < "/proc/meminfo";*  *#CPU percentage is the 2nd parameter of this function*  *echo "CPU $2"*  *#And # of process is the first one*  *echo -n "Procesos activos : $1 "*  *#Since the data have another fields, we need to filter them.*  *totalMemory=$(echo "${memData[0]}" | sed 's/[^0-9]\*//g')*  *freeMemory=$(echo "${memData[1]}" | sed 's/[^0-9]\*//g')*  *#We can express usedMemory as the diff of total and free mem*  *usedMemory=$(expr $totalMemory - $freeMemory)*  *echo -n " -- Memoria total= $totalMemory kb"*  *echo -n " -- Memoria libre= $freeMemory kb"*  *echo " -- Memoria usada= $usedMemory kb"*  *echo*  *}*  *function printProcessData {*  *#ls will retrieve all the directories, but we need to use grep in order to filter*  *#only the numeric ones.*    *read utime idltime < "/proc/uptime"*      *read cpu a b c d e f g h i j< "/proc/stat"*    *let "firstTotal= $a + $b + $c + $d + $e + $f + $g + $h + $i + $j"*    *processCommand=$(ls /proc -t | grep '^[0-9]\*$');*  *#And then we will save it in an array*  *proccessArray=();*    *#Total memory will be used for Mem percentage calcs.*  *readarray memData < "/proc/meminfo";*  *totalMemory=$(echo "${memData[0]}" | sed 's/[^0-9]\*//g')*    *#For each process we need to validate that it*  *#exists and the we append it to processArray*  *for i in $processCommand; do*  *if [ -d "/proc/${i%%/}" ]; then*  *processArray+=("${i%%/}");*  *fi*  *done*    *#totalTimes1() is the first read of CPU times*  *totalTimes1=()*    *echo "Primer lectura de tiempos"*    *#Then we need to fill totalTimes1() and wait 1s*  *for i in ${processArray[@]}; do*    *readarray lines < "/proc/$i/stat";*    *statData=();*  *for x in ${lines[@]}; do*  *statData+=("$x")*  *done*  *totalTime=0*    *let " totalTime= ${statData[13]} + ${statData[14]}"*    *totalTimes1+=("$totalTime")*  *done*  *sleep 1*    *read cpu a b c d e f g h i j < "/proc/stat"*    *let "secondTotal= $a + $b + $c + $d + $e + $f + $g + $h + $i + $j"*    *let "diffTime = $secondTotal - $firstTotal"*    *j=0*    *clear*    *echo "Segunda lectura de tiempos y creacion de pidData.txt..."*      *#Total CPU is the summatory of each PID's CPU percentage*  *totalCPU=0*    *for i in ${processArray[@]}; do*    *readarray lines < "/proc/$i/stat";*    *statData=();*  *for x in ${lines[@]}; do*  *statData+=("$x")*  *done*  *totalTime=0*    *let " totalTime= ${statData[13]} + ${statData[14]}"*      *#Getting the diff of totalTime after sleep 1s*    *let "cpuPercentage = $totalTime - ${totalTimes1[j]}"*      *let "cpuPercentage = 100 \* $cpuPercentage / $diffTime "*      *let "totalCPU = $cpuPercentage + $totalCPU"*      *#Memory PID is expressed in bytes, so we need to convert it to kb*    *let "memoryPID = ${statData[22]} / 1024 "*    *#And with total mem we get a %*    *let "memPercentage = 100 \* $memoryPID / $totalMemory "*      *#getting username through user id*  *uid=$(awk '/^Uid:/{print $2}' /proc/"$i/"status)*  *uname=$(getent passwd "$uid" | awk -F: '{printf $1 }')*  *file="pidData.txt"*    *#The information is redirected to a file, so we can use sort it*  *echo -e "${statData[0]} \t ${statData[17]} \t\t ${statData[2]} \t $cpuPercentage\t $memPercentage \t $totalTime \t\t ${statData[1]} \t\t "$uname" \t\t $memoryPID " >> $file*    *let "j= $j + 1"*  *done*    *clear*    *if [ $totalCPU -gt 100 ] # integer ops can make $totalCPU reach a value gt 100, so we adjust it*    *then*  *let "totalCPU = 100"*  *fi*  *#In the standard output we print header data*  *printHeaderData "${#processArray[@]}" "$totalCPU";*    *echo -e "PID \t Prior \t\t Estado %CPU \t %MEM \t Tiempo \t Commando \t\t Usuario \t Memoria "*  *echo*  *}*  *#end auxiliar functions section*  *#Cleaning auxiliar file*  *echo "Limpiando archivo pidData.txt"*  *truncate -s 0 "pidData.txt"*  *clear*  *printProcessData*  *#We use this in order to sort information by %CPU in reverse order.*  *sort -nrk 4 pidData.txt | head -n 10* |

## Ejecución de ejemplo

Para la primera prueba se ejecutó el programa mediante Ejercicio1.sh, esperando que el proceso yes apareciese en la parte alta de mitop:



Esto puede comprobarse en este ejemplo, en el cual el proceso yes aparece en la parte alta de mitop con el PID 3780.

Del ejemplo de ejecución anterior también se puede apreciar que los procesos son ordenados de mayor a menor de acuerdo a su uso de procesador, por razones de precisión y la falta de una opción nativa de operaciones con punto flotante en bash los porcentajes son dados en valores enteros, de este modo algunos valores se verán redondeados y en algunos casos truncados a 0.



En contraparte, una segunda ejecución directa del script (sin la creación del proceso yes), muestra resultados diferentes con un uso de CPU moderado en comparación al anterior, siendo esta vez el script de bash el que encabeza la lista de uso de CPU, y reduciéndose el número de procesos activos como se puede ver en el encabezado, la memoria total a modo de comprobación no ha cambiado por obvias razones, pero no es así con la memoria libre y utilizada.

Al igual que en la ejecución anterior, valores de uso de CPU en el rango de 0 a 1 pueden verse truncados debido al método usado para obtener los valores, mismo que se discute a continuación en el siguiente apartado.

## Justificación de uso de let como método para operaciones aritméticas

Para realizar el algoritmo de cálculos de porcentajes en el programa presentado se tuvo a bien investigar sobre las diversas maneras de realizar operaciones aritméticas en bash, de esta manera se encontraron varios métodos con una separación notoria; aquellos que permiten operaciones con punto flotante y los que no la permiten.

La razón de haber elegido **let** por sobre otros métodos como llamados a **bc** reside en un objetivo específico: lograr que el tiempo de procesamiento en cada iteración y por ende el tiempo de lectura de los archivos de /proc/PID fuera el mínimo.

Los métodos de operaciones con punto flotante no son nativos de bash, por lo que su uso requeriría una llamada a un programa externo, lo cual para nuestro propósito es ineficiente y una opción a descartar, durante las pruebas de las versiones del programa que se realizaron con operaciones que soportaban punto decimal la ejecución del programa dejaba mucho que desear siendo de varios segundos, mismos que al tratarse de recuperar información de archivos de procedimientos conllevaban a tener cambios de información y datos imprecisos.

Es por esto que se eligió let por sobre otras opciones, aun sacrificando precisión en resultados la velocidad con la que se ejecuta el script es por demás aceptable y los resultados de tener procesos con tiempos más cortos son más precisos.

# Bibliografía

|  |  |
| --- | --- |
| [1] | L. org, «helpmanual.io,» Linux, [En línea]. Available: https://helpmanual.io/builtin/readarray/. [Último acceso: 2018 Octubre 20]. |
| [2] | L. org, «helpmanual.io,» [En línea]. Available: https://helpmanual.io/man1/grep/. [Último acceso: 20 Octubre 2018]. |
| [3] | «ss64,» [En línea]. Available: https://ss64.com/bash/bc.html. [Último acceso: 20 Octubre 2018]. |
| [4] | «Geeks for Geeks,» [En línea]. Available: https://www.geeksforgeeks.org/sort-command-linuxunix-examples/. [Último acceso: 20 Octubre 2018]. |