DISEÑO Y ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

PED1

SHELL SCRIPT 'MITOP.SH'

Misrraim Suárez Pérez - misrraimsp@gmail.com

Introducción

En este trabajo se ha desarrollado un bash script que imita, muy humildemente, al comando *top*. Este script se llama *mitop*, y presenta por pantalla diferente información estadística de los procesos con mayor uso de la CPU.

Implementación

Desde un punto de vista global el script se divide, tal como está documentado en el propio código, en siete bloques funcionales. Estos bloques se ejecutan (salvo la parte de definición de funciones) secuencialmente. A continuación se describe cada uno de ellos:

1. Configuración

   Lo primero que se hace es definir una serie de constantes usadas durante el resto del script.

   La primera de ellas, *hrtz,* es el factor de conversión entre ticks del sistema y segundos.

   Las tres siguientes configuran parte de la apariencia de la salida por pantalla. Los vectores *ancho_campos* y *ancho_cabecera* definen el espacio asignado a cada columna de, respectivamente, los campos y la cabecera. La constante *brick* es el elemento con que se construye las líneas divisoras.

2. Funciones

   Después defino tres funciones de utilidad en el resto del código: *pid2uname()*, *format()* y *getline()*.

   pid2uname() - Esta función recibe como parámetro el PID de un proceso dado. Con ese valor accede (si puede) al fichero '/proc/PID/status' para obtener el UID asociado al proceso. Con el identificador de usuario accede al fichero '/etc/passwd' y filtra el nombre de usuario buscado.

   format() - Esta función espera dos parámetros. El primero una cadena de caracteres. El segundo un valor entero. Como salida proporciona la misma cadena que la entrada pero añadiendo espacios (o suprimiendo caracteres) del final, de forma que la cadena de salida tenga la longitud especificada en el segundo parámetro.

   getline() - Esta función se encarga de producir una cadena de caracteres, usando siempre el mismo carácter (pasado como su único parámetro), de forma que su longitud sea coincidente con la suma de los anchos de campo definidos en el vector de configuración *ancho_campos*.

3. Lecturas

En esta fase se accede a los ficheros en '/proc' necesarios para leer la información necesaria para los cálculos de uso de la CPU. Se ha minimizado sus operaciones con objeto de maximizar la velocidad de ejecución y obtener resultados confiables.

Primero se obtiene la lista de los procesos en el sistema, ordenada. Después se realiza la secuencia:

(1) leer el contenido de '/proc/stat' y almacenarlo en *SyStat_i*

(2) para cada proceso, leer (si existe) el contenido de '/proc/PID/stat' y almacenarlo en *PrStat_i[PID]*

(3) dormir 1 segundo

(4) leer el contenido de '/proc/stat' y almacenarlo en *SyStat_f*

(5) para cada proceso, leer (si existe) el contenido de '/proc/PID/stat' y almacenarlo en *PrStat_f[PID]*

4. Uso total de la CPU

Los diferentes números de ticks de CPU desde el arranque de la máquina han sido obtenidos de '/proc/stat' en la fase de lectura y almacenados en *SyStat_i* y *SyStat_f*. A partir de dicha información se realiza el cálculo del uso total de la CPU.

Un ejemplo del contenido de las variables *SyStat_i* y *SyStat_f* tras la lectura se muestra a continuación:

```
$ cat /proc/stat
cpu  4705 356  584    3699   23    23     0      0     0         0
```

El significado de cada campo se entiende desde [1]. Teniendo en cuenta [2], el %CPU total se obtiene como:

total_i = user_i + nice_i + system_i + idle_i + iowait_i + irq_i + softirq_i + steal_i

total_f = user_f + nice_f + system_f + idle_f + iowait_f + irq_f + softirq_f + steal_f

idle_i = idle_i + iowait_i

idle_f = idle_f + iowait_f

idle_diff = idle_f – idle_i

total_diff = total_f – total_i

$$\%CPU = \left(1 - \frac{idle_{diff}}{total_{diff}}\right) * 100$$

Notar que *guest* y *guest_nice* son contabilizados, respectivamente, dentro de *user* y *nice*, de ahí que no sean incluidos en el cálculo.

5. Uso total de la memoria

   En esta fase se obtienen los datos de memoria del fichero '/proc/meminfo'. El primer campo corresponde con la memoria total, mientras que el segundo con la libre. Por diferencia de los anteriores se obtiene la memoria ocupada.

6. Gestión de procesos

   Una vez obtenida la información global del sistema (%CPU y memoria) para completar la cabecera, se procede a tratar individualmente con los procesos.

   Esta fase está funcionalmente dividida en dos partes:

   (1) Para cada proceso de la lista de procesos que tengan contenido almacenado en ambos vectores *PrStat_i* y *PrStat_f* se extraen los campos 14 y 15. Estos campos representan el tiempo de uso de la CPU en modo usuario y modo kernel. Con estos tiempos se calcula el incremento de tiempo, *inc*, y se conforma el contenido de una variable auxiliar, *todos,* como una secuencia "*inc*|*tPID*|*ninc*|*tPID*|*n...inc*|*tPID*|*n".* Esto permite ordenar fácilmente de forma decreciente según el campo *inc*, y confeccionar la variable de entrada para la siguiente fase, *filtrados*, con los procesos de mayor *inc*. Se ha dejado abierto al usuario el número de procesos a seleccionar como parámetro del script, aunque si no se especifica se escogen los 10 primeros. Es en esta fase también donde se obtiene el número de procesos del sistema.

   (2) La variable *filtrados* también es de la forma "*inc*|*tPID*|*ninc*|*tPID*|*n...inc*|*tPID*|*n",* pero ya conteniendo, ordenados, los procesos seleccionados. Es en esta fase cuando, para cada uno de dichos procesos, se da forma al resto de información a presentar. Esto se hace recorriendo el segundo campo de la variable *filtrados* y concatenando las diferentes informaciones obtenidas de *PrStat_i[PID]* en la variable *salida*. Cada iteración sobre los procesos seleccionados conforma una línea de la variable *salida.* Notar que es aquí donde se hace uso de la función *format()* para garantizar que los campos se ajustan a los anchos definidos en el vector *ancho_campos*. Notar también que el campo NAME se obtiene con una llamada a la función *pid2uname()*.

7. Impresión

   Lo único que resta llegados a este punto es sacar por pantalla la información recolectada. Primero se llama a la función *getline()* para obtener la línea delimitadora. Después se escribe la cabecera, haciendo uso de *format()* con los anchos definidos en el vector *ancho_cabecera*. Lo siguiente es emprimir en pantalla la leyenda de los campos de información de cada proceso. Por último se saca por pantalla el contenido de la variable *salida*, con toda la información formateada de los procesos seleccionados.

Referencias

[1] http://man7.org/linux/man-pages/man5/proc.5.html

[2] https://github.com/Leo-G/DevopsWiki/wiki/How-Linux-CPU-Usage-Time-and-Percentage-is-calculated#overview

[3] http://stackoverflow.com/questions/16726779/how-do-i-get-the-total-cpu-usage-of-an-application-from-proc-pid-stat

[4] **Learning the bash Shell**, 3rd Edition. Cameron Newham

[5] http://wiki.bash-hackers.org/start