

# Lab1: Observando el Comportamiento de Linux (2016)

## Objetivos

Iniciar los laboratorios con un proyecto sencillo, donde se muestre como a través del sistema de archivos virtual **/proc** de Linux, podemos inspeccionar información interna del kernel. Se deberá generar la utilidad **ksamp** que muestra de diversas formas algún subconjunto de la información disponible en **/proc**.

## Introducción

Podemos pensar el **kernel de Linux** como una colección de funciones y estructuras de datos. Estas estructuras de datos (o variables de kernel) contienen la visión del kernel respecto al estado del sistema, donde cada interrupción, cada llamada al sistema, cada fallo de protección hacen que este estado cambie.

Inspeccionando las variables del kernel podemos obtener información relevante a los procesos, interrupciones, dispositivos, sistemas de archivos, capacidades del hardware, etc. Muchas variables conforman el estado del kernel y estas pueden estar alojadas de manera estática o dinámica en un stack frame de la memoria del kernel. Dado que el kernel de Linux es un programa "C" de **código abierto**, es posible inspeccionar el código fuente y encontrar algunos ejemplos relevantes de estas variables. Por ejemplo, en kernels 2.4.x, `xtime` definido en `include/linux/sched.h` que mantiene la hora del sistema, la estructura `task_struct` definida en `include/linux/sched.h` que contiene la descripción completa de un proceso, o los valores `nr_threads` y `nr_running` definidos en `kernel/fork.c` los cuales indican cuantos procesos existen y cuántos de estos están corriendo.

El código fuente de Linux lo podemos encontrar en el directorio `/usr/src/linux` de la mayoría de las distribuciones, pero también existen páginas donde podemos navegar el código de varias versiones de kernel, como por ejemplo [Linux Cross Reference](#). También existen utilidades para navegación de código local como [Source Navigator](#).

## Tareas

### Parte A

Buscar información acerca estructura del directorio **/proc**, y averiguar los siguientes datos, teniendo en cuenta la versión de kernel que está corriendo:

- Tipo y modelo de CPU.
- Versión del kernel.

- Tiempo en días, horas, minutos y segundos que han transcurrido desde que se inició el sistema operativo.
- Cuánto tiempo de CPU ha sido empleado para procesos de usuario, de sistema y cuando tiempo no se usó.
- Cuánta memoria tiene y cuánta está disponible.
- Cuántos sistemas de archivo soporta el kernel
- Cuántos pedidos de lectura/escritura a disco se han realizado.
- Cuántos cambios de contexto han sucedido.
- Cuántos procesos se crearon desde que inició el sistema.

### **Ejemplo:**

Type: GenuineIntel  
 Model: Intel(R) Xeon(TM) CPU 2.40GHz  
 Kernel: 2.4.18-custom.1.3  
 UpTime: 1D 5:32:47.43  
 UserTime: 1:49.9  
 SysTime: 6:55.32  
 IdleTime: 1D 5:24:03.02  
 TotalMem: 922587136  
 FreeMem: 15532032  
 Nbr of FS: 29  
 Disk: 667810  
 Context: 16336100  
 Processes: 18224

## **Parte B**

Escriba una versión inicial del ksamp programa que inspeccione las variables del kernel a través del /proc e informe por stdout:

- Tipo y modelo de CPU.
- Versión de Kernel.
- Cantidad de tiempo transcurrido desde que se inició el sistema operativo, con el formato ddD hh:mm:ss.
- Cantidad de sistemas de archivo soportados por el kernel

También se pide incluir una cabecera donde se indique el nombre de la máquina y la fecha y hora actuales.

## **Parte C**

Escriba una segunda versión del programa que imprima la misma información que la versión por defecto, pero en caso de invocarse con la opción -s, agrega la siguiente información:

- Cantidad de tiempo de CPU utilizado para usuarios, sistema y proceso idle.

- Cantidad de cambios de contexto.
- Fecha y hora cuando el sistema fue iniciado.
- Número de procesos creados desde el inicio del sistema.

## Parte D

La tercera parte involucra imprimir todo lo de las versiones anteriores, pero cuando se invoca con la opción `-l interval duration` imprime además:

- Número de peticiones a disco realizadas.
- Cantidad de memoria configurada en el hardware.
- Cantidad de memoria disponible.
- Lista de los promedios de carga de 1 minuto.

Así por ejemplo `ksamp -l 2 100` mostrará el promedio de carga de 1 minuto por 100 segundos tomando muestras en intervalos de 2 segundos. El comando `ksamp -l a b`, lee los datos indicados arriba (peticiones de disco, cantidades de memoria, promedios de carga) de `/proc`, y los imprime repetidamente cada `a` segundos; esto se repite hasta que hayan pasado `b` segundos. Por ejemplo:

```
$ ksamp -l 5 15
Peticiones a disco: 12345
Memoria disponible / total: 1000000 / 8000000
Promedio de carga en el último minuto: 0.88
[Pausa de 5 segundos]
Peticiones a disco: 12348
Memoria disponible / total: 2000000 / 8000000
Promedio de carga en el último minuto: 0.82
[Pausa de 5 segundos]
Peticiones a disco: 12645
Memoria disponible / total: 500000 / 8000000
Promedio de carga en el último minuto: 0.98
[Pausa de 5 segundos]
Peticiones a disco: 99995
Memoria disponible / total: 300000 / 8000000
Promedio de carga en el último minuto: 1.07
[so2016@hal so2016]$
```

En particular noten que el promedio de carga de un minuto es un valor que el sistema provee ya calculado, no es algo que ustedes tengan que calcular.

Notar que cada opción incluye a la otra, por lo que `ksamp -s -l 2 100` no debería ser aceptado y en tales casos resulta útil imprimir por la salida estándar un resumen de las opciones aceptadas.

# Cómo atacar los problemas

Se recomienda utilizar el libro *Advanced Linux Programming*. Para este práctico, en particular el capítulo 7: "The /proc File System."

La página del manual de Linux man 5 proc contiene suficiente información al respecto, también se pueden ver artículos de The Official Red Hat Linux Reference Guide en su capítulo "The /proc Filesystem" . En general basta con realizar una búsqueda de "/proc filesystem" en cualquier buscador de la Web para encontrar información en cualquier idioma.

Una vez encontrados los archivos de /proc donde está la información, es necesario abrirlos, leer la información y cerrarlos. Esto se puede lograr con las funciones de la biblioteca "C" fopen, fgets (o fscanf) y fclose. Un ejemplo de uso sería el siguiente:

```
#include <stdio.h>
#define BUFFSIZE 256
int
main(int argc, char *argv[]) {
    FILE *fd;
    char buffer[BUFFSIZE+1];
    fd = fopen("/proc/sys/kernel/hostname", "r");
    fgets(buffer, BUFFSIZE+1, fd);
    printf("Hostname: %s\n", buffer);
    fclose(fd);
}
```

Para leer los argumentos de entrada e interpretarlos (proceso de parsing) hay que hacer uso de las variables int argc y char \*argv[]. La primera indica cuantos argumentos se pasaron y argv es el arreglo de tamaño argc con cada uno de las cadenas de los argumentos. Notar que como el propio comando se incluye en la lista de argumentos una llamada ksamp -l 10 100 implica que al inicio del main se cumple

```
{argc=4 & argv={"ksamp", "-l", "10", "100"}}.
```

Pueden utilizar la función getopt de la glibc para no volver a inventar la rueda.

Pueden encontrar información en man 3 getopt o bien en las Infopages invocando pinfo libc y buscado este tema. Este último comando muestra la GNU C Library reference manual .

El ejemplo anterior resulta sencillo en cuanto a que no tenemos que "interpretar" la secuencia de caracteres, sólo la tomamos y la imprimimos. Sin embargo, muchos de los parámetros que se necesitan imprimir requieren de cierto tratamiento. Este proceso de transformar una secuencia de caracteres en alguna representación más adecuada para su tratamiento se denomina parsing. Por ejemplo el tiempo transcurrido desde que el sistema

inició se expresa en segundos, y se encuentra en cierta parte del archivo. Entonces tenemos que extraerlo y convertirlo a un entero sin signo. Cuando ya tenemos un entero sin signo, resulta sencillo operar matemáticamente y generar una secuencia de caracteres (imprimir) con el formato adecuado. Esto último se denomina **pretty printing** . Funciones como atoi y sus relacionadas pueden ser útiles. También resulta útil sscanf. Para obtener la fecha y hora que va en el encabezado del informe que brinda el programa, se pueden utilizar las funciones de la glibc gettimeofday y ctime, o bien leer el tiempo de inicio del sistema y sumarlo al transcurrido para luego convertirlo a fecha.

Cuando se necesite realizar la muestra del promedio de carga de 1 minuto, puede ser útil la función sleep que duerme un proceso por un intervalo determinado de tiempo.

## Qué se debe Entregar

Código comentado (funcionando bajo las especificaciones dadas y bajo cualquier caso de test de parámetros de entrada).

Utilizar flags Wall y Pedantic para compilar el proyecto.

El código no debe contener ni errores ni warnings, luego de correr el analizador de código estático Cppcheck

Dividir en módulos de manera juiciosa.

Estilo de código: GNU Coding Standard for C

Utilizar Makefile.

Informe del desarrollo del proyecto.

## Tips

No intenten hacer todo de golpe, vayan de a partes, y sobre todo discutan, analicen y trabajen en ideas de forma grupal. Una hora pensando en papel libremente suele ahorrar muchos problemas en el momento de codificar.

Utilicen debuggers de línea de comandos como gdb o interfaces gráficas para estos como ddd. También pueden realizar compilación condicional para hacer debugging de la siguiente manera:

```
#define STDOUT 0
#define DEBUG

.
.
#ifdef DEBUG
fprintf(STDOUT,"DEBUG> argc: %d\n",argc);
#endif
.
.
```

Deducir la entrada `disk_io` requiere un poco más de trabajo que el resto de la información. Encuentren el manejo de `/proc` en kernels 2.4.x y 2.6.x, analicen el código fuente y deduzcan el significado de cada elemento. Pueden utilizar [Linux Cross Reference](#).

Para comentar el código pueden utilizar Doxygen ([www.doxygen.org](http://www.doxygen.org)). Desde la web oficial se puede descargar la GUI llamada Doxywizard para la configuración del proyecto.

Para el desarrollo del informe se puede generar un archivo README, utilizando el lenguaje de marcado ligero Markdown.

## Tareas Adicionales

Se pueden realizar las siguientes mejoras:

- Algunos de los datos que se muestran involucran cantidades de memoria, y el valor que se puede leer en `/proc` está siempre expresado en bytes. Como leer cantidades como 64MB en bytes suele ser dificultoso, muchos programas incluyen la opción `-h` que significa "legible para humanos", que transforma las cantidades a una unidad correspondiente de manera que la

cadena se mantenga acotada en tamaño.

- Este programa puede ser muy útil para monitorear la actividad de un servidor, pero resulta más práctico si se le agrega una opción `-d` que bajo la opción `-l` muestra el diferencial de los valores leídos para, por ejemplo, poder medir cuántos cambios de contexto se están realizando por segundo.

Este es un muy buen lugar para utilizar un TAD con los valores de muestra que acepte métodos que calculen el diferencial entre muestras.

- Utilizar `getopt` de POSIX.2 para parsear la línea de comandos.

- Comparar lo realizado con utilidades establecidas que realizan tareas similares, como por ejemplo `procinfo`.