

Laboratorio #4
Sistemas Operativos
2021

Alumno: **Estuardo Ureta 17010**

Fecha de Entrega: 28 de marzo, 2021.

Descripción: este laboratorio reforzará sus conocimientos de diseño e implementación de sistemas operativos con tres ejercicios: creación y carga de un módulo propio al *kernel*; uso de la herramienta SystemTap; e instalación de un *bootstrap program* llamado LILO. Debe entregar en Canvas un archivo de texto con sus respuestas a las preguntas planteadas y con las capturas de pantalla solicitadas.

Materiales: se recomienda la máquina virtual OSC-2016 para el ejercicio 2, aunque la instalación y remoción de un módulo por medio de un programa debería ser logable en versiones más recientes de Linux con instrucciones similares.

El ejercicio 3 requiere reemplazar el sistema de arranque GRUB por el sistema de arranque LILO. Las instrucciones garantizan esta meta si se trabaja sobre la máquina OSC-2016. De trabajarse en otro sabor o versión de Linux, el objetivo debe ser instalar LILO **manualmente**, por lo que debería dejarse registro de los pasos tomados, principalmente de las diferencias que haya con respecto a las instrucciones presentadas en este documento.

El ejercicio 1 puede desarrollarse en cualquier sistema Linux mientras se pueda instalar SystemTap.

Contenido

Ejercicio 1 (30 puntos)

- a. Descargue la herramienta SystemTap con el siguiente comando:

```
sudo apt-get install systemtap
```

- b. Cree un archivo llamado `profiler.stp`, con el siguiente código:

```
probe timer.profile{  
    printf("Proceso: %s\n", execname())  
    printf("ID del proceso: %d\n", pid()) }  
}
```

- c. Ejecute su archivo usando el siguiente comando:

```
sudo stap profiler.stp
```

- ¿Qué puede ver en el *output* cuando realiza estas acciones?

```

oscreader@OSC: ~
File Edit View Search Terminal Help

Proceso: Xorg
ID del proceso: 720
Proceso: Xorg
ID del proceso: 720
Proceso: Xorg
ID del proceso: 720
Proceso: gnome-shell
ID del proceso: 1118
Proceso: Xorg
ID del proceso: 720
Proceso: gnome-shell
ID del proceso: 1118
Proceso: gnome-shell
ID del proceso: 1118
Proceso: gnome-shell
ID del proceso: 1118
Proceso: gnome-shell
ID del proceso: 1118
Proceso: gnome-shell
ID del proceso: 1118
Proceso: Xorg
ID del proceso: 720

```

- ¿Para qué sirve SystemTap?

- ¿Qué es una *probe*?

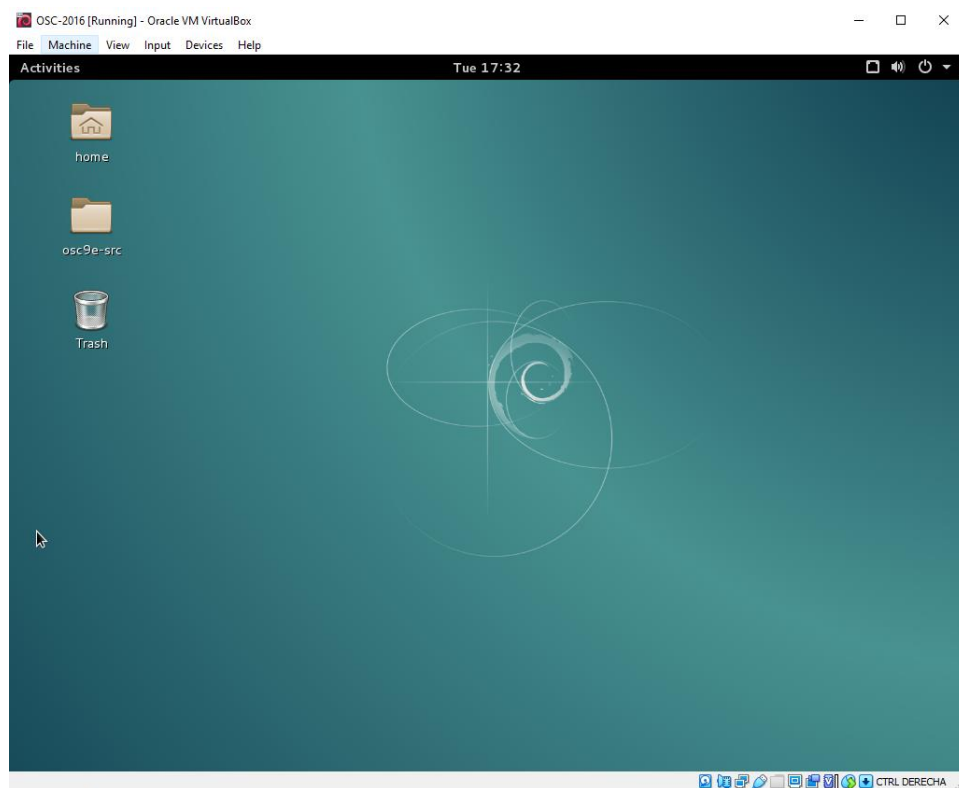
- **¿Cómo funciona SystemTap?**

- ¿Qué es hacer *profiling* y qué tipo de *profiling* se hace en este ejercicio?

No es más que una forma de análisis de rendimiento y comportamiento que mide cosa como memoria o la complejidad temporal de un programa. Los scripts del ejercicio son scripts que pueden diseñarse para extraer datos, filtrarlos y resumirlos de forma rápida, lo que permite el diagnóstico de problemas complejos de rendimiento.

Ejercicio 2 (30 puntos)

- a. Abra su máquina virtual y tómela una *snapshot*.



- b. Cree un programa en C llamado `simple.c`. Este programa deberá #incluir los siguientes encabezados:
- `<linux/init.h>`
 - `<linux/kernel.h>`
 - `<linux/module.h>`
 - `<linux/list.h>`
- c. Escriba dos métodos en su programa llamados `simple_init` y `simple_exit`. Ambos métodos deben declarar como parámetro únicamente `void`, y el primero debe retornar tipo `int` mientras que el segundo tipo `void`. El primer método debe devolver cero.
- ¿Cuál es la diferencia en C entre un método que no recibe parámetros y uno que recibe `void`?
- d. En el primer método incluya la siguiente instrucción:

```
printk(KERN_INFO "Loading Module\nSistops");
```

Reemplace el texto `Sistops` por un mensaje personalizado. En el segundo incluya la siguiente instrucción:

```
printk(KERN_INFO "Removing Module\nSistops");
```

Nuevamente reemplace el texto `Sistops` por un mensaje personalizado.

- **¿Qué diferencia hay entre `printk` y `printf`?**

`printk` es una función de nivel de kernel, que tiene la capacidad de imprimir en diferentes niveles de registro como se define en `<linux/kernel>`. La diferencia es que el `printk` tiene la capacidad de especificar un nivel de log.

- **¿Qué es y para qué sirve `KERN_INFO`?**

Es un nivel de log que nos permite mostrar mensajes críticos o estados del sistema. `KERN_INFO` es el sexto de estos niveles y este es el nivel de registro utilizado para mensajes informativos sobre una acción realizada por el kernel.

- e. Abajo de sus dos métodos incluya las siguientes instrucciones (reemplazando `<Su nombre>` con su nombre y `<Descripcion>` con una descripción personalizada):

```
module_init(simple_init); module_exit(simple_exit);  
MODULE_LICENSE("GPL");  
MODULE_DESCRIPTION("<Descripcion>");  
MODULE_AUTHOR("<Su nombre>");
```

Grabe su programa.

- f. Cree un archivo *Makefile* para su programa, que contenga el siguiente código:

```
obj-m += simple.o  
  
all:  
    make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd)  
modules clean:  
    make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

- **¿Qué es una *goal definition* o definición de meta en un *Makefile*, y qué se está haciendo con la definición de meta `obj-m`?**

Básicamente, una *goal definition* no es más que el objetivo que el comando *make* debería enfocarse en, para su actualización. El sistema *kbuild* en el kernel construirá *simple.o* desde *simple.c*. Después de vincular estos archivos, obtendrá el módulo del *simple.ko*.

- **¿Qué función tienen las líneas *all* : y *clean* : ?**

All denota que, si lo invoca, *make* compilará todo lo que se necesita para hacer una compilación completa. *Clean* elimina todos los archivos de objeto que se han creado mientras tanto.

- **¿Qué hace la opción *-C* en este *Makefile*?**

Cambia el directorio antes de leer o actualizar los programas.

- **¿Qué hace la opción *M* en este *Makefile*?**

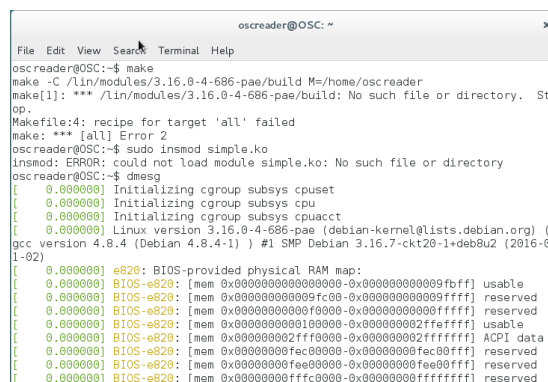
Especifica en que directorio está presente el módulo.

g. Ejecute el comando *make* en el directorio donde haya creado *simple.c* y su correspondiente *Makefile*.

h. Ejecute los siguientes comandos:

```
sudo insmod simple.ko dmesg
```

Tome una captura de pantalla de los resultados de ambos comandos e inclúyala en sus entregables.



```
oscreader@OSC: ~  
File Edit View Search Terminal Help  
oscreader@OSC:~$ make  
make -C /lin/modules/3.16.0-4-686-pae/build M=/home/oscreader  
make[1]: *** /lin/modules/3.16.0-4-686-pae/build: No such file or directory. St  
op.  
Makefile:4: recipe for target 'all' failed  
make: *** [all] Error 2  
oscreader@OSC:~$ sudo insmod simple.ko  
insmod: ERROR: could not load module simple.ko: No such file or directory  
oscreader@OSC:~$ dmesg  
[ 0.000000] Initializing cgroup subsys cpuset  
[ 0.000000] Initializing cgroup subsys cpu  
[ 0.000000] Initializing cgroup subsys cpuacct  
[ 0.000000] Linux version 3.16.0-4-686-pae (debian-kernel@lists.debian.org) (g  
cc version 4.8.4 (Debian 4.8.4-1) ) #1 SMP Debian 3.16.7-ckt20-1+deb8u2 (2016-0  
1-02)  
[ 0.000000] e820: BIOS-provided physical RAM map:  
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable  
[ 0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x00000000000a0000-0x00000000000affff] reserved  
[ 0.000000] BIOS-e820: [mem 0x00000000000b0000-0x00000000000bffff] usable  
[ 0.000000] BIOS-e820: [mem 0x00000000000c0000-0x00000000000cffff] ACPI data  
[ 0.000000] BIOS-e820: [mem 0x00000000000d0000-0x00000000000dffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x00000000000e0000-0x00000000000effff] reserved  
[ 0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
```

- **¿Para qué sirve *dmesg*?**

Sirve para supervisar el Kernel Ring Buffer para así imprimir los mensajes de inicio del módulo.

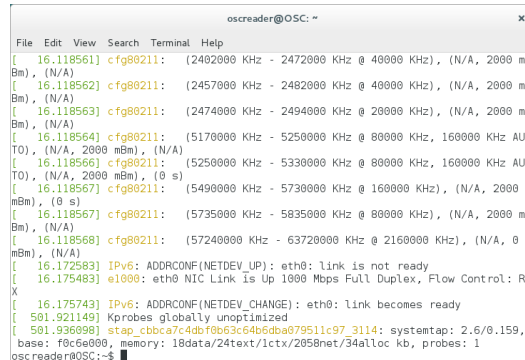
- **¿Qué hace la función *simple_init* en su programa *simple.c*?**

Empieza el módulo e imprime el proceso encargado del mismo.

- i. Ahora ejecute los siguientes comandos:

```
sudo rmmod simple dmesg
```

Tome una nueva captura de pantalla de los resultados de ambos comandos e inclúyala en sus entregables.



```
oscreader@OSC: ~
File Edit View Search Terminal Help
[ 16.118561] cfg80211: (2482000 KHz - 2472000 KHz @ 40000 KHz), (N/A, 2000 m
Bm), (N/A)
[ 16.118562] cfg80211: (2457000 KHz - 2482000 KHz @ 40000 KHz), (N/A, 2000 m
Bm), (N/A)
[ 16.118563] cfg80211: (2474000 KHz - 2494000 KHz @ 20000 KHz), (N/A, 2000 m
Bm), (N/A)
[ 16.118564] cfg80211: (5170000 KHz - 5250000 KHz @ 80000 KHz, 160000 KHz AU
TO), (N/A, 2000 mBm), (N/A)
[ 16.118566] cfg80211: (5250000 KHz - 5330000 KHz @ 80000 KHz, 160000 KHz AU
TO), (N/A, 2000 mBm), (0 s)
[ 16.118567] cfg80211: (5490000 KHz - 5730000 KHz @ 160000 KHz), (N/A, 2000
mBm), (0 s)
[ 16.118567] cfg80211: (5735000 KHz - 5835000 KHz @ 80000 KHz), (N/A, 2000 m
Bm), (N/A)
[ 16.118568] cfg80211: (57240000 KHz - 63720000 KHz @ 2160000 KHz), (N/A, 0
mBm), (N/A)
[ 16.172583] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[ 16.175483] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: R
X
[ 16.175743] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 501.921149] Kprobes globally unoptimized
[ 501.936090] stcp_c0bca7c4dbf0b53c64b6db979511c97_3114: systemtap: 2.6/0.159,
base: f0c6e000, memory: 18data/24text/1ctx/2058net/34alloc kb, probes: 1
oscreader@OSC:~$
```

- ¿Qué hace la función `simple_exit` en su programa `simple.c`?

Esta función se llama cuando se elimina el módulo del Kernel.

- Usted ha logrado crear, cargar y descargar un módulo de Linux. ¿Qué poder otorga el ejecutar código de esta forma?

Principalmente el control de que módulos se utilizaran, así como saber las dependencias de nuestros programas para no incurrir en errores de directorios u otros. También de esta manera, evitamos redundancia ya que podemos reutilizar el código en nuestros programas.

Ejercicio 3 (40 puntos)

- a. Si todo ha salido bien con los demás ejercicios, tómese una *snapshot* a su máquina virtual. De lo contrario no continúe con este ejercicio y complete los demás, asegurándose de que su sistema queda estable. Repito: **no continúe este ejercicio sin sacar una *snapshot* estable de su máquina primero.**
- b. Ejecute el siguiente comando en una terminal (note el guion al final):

```
sudo apt-get --purge install lilo grub-legacy-
```

Durante la instalación aparecerá una pantalla que le indicará ejecutar `liloconfig` y `/sbin/lilo` más adelante. Presione *Enter* e ignórela. Estos comandos harían automáticamente lo que los siguientes incisos le ayudarán a hacer “a pie”.

- c. Vaya al directorio `/dev/disk/by-id` y ejecute el comando `ls -Al`. El resultado le mostrará varios *links* simbólicos, algunos de los cuales se dirigen a algo igual o parecido a `../../sda`. Anote el nombre del *link* que no incluye algo como `"partN"` y que apunta exactamente a `../../sda`.
- d. Vaya al directorio `/etc` y lea el contenido del archivo `fstab`. Verá una tabla (probablemente desalineada) y deberá buscar la fila cuya columna llamada `<mount point>` contenga `"/"`. De esa fila anote el contenido de la columna `<file system>`.

- **¿Qué es y para qué sirve el archivo `fstab`?**

Es un archivo que tiene descriptores de la máquina que monta frecuentemente. Sirve para montar filesystems al realizar un *systemboot*.

- **¿Qué almacena el directorio `/etc`? ¿En Windows, quién (hasta cierto punto) funge como `/etc`?**

Almacena todos los archivos de configuración del sistema. En Windows hay un archivo llamado `hosts` que se encuentra dentro de la carpeta `/etc` y este es el que desempeña el trabajo de manera temporal.

- **¿Qué se almacena en `/dev` y en `/dev/disk`?**

En `/dev` se almacenan archivos de dispositivo especiales para todos los dispositivos mientras que en `/dev/disk` se almacena info. sobre las particiones del sistema.

- e. En ese mismo directorio `/etc` cree un archivo llamado `lilo.conf` que contenga lo siguiente:

```
boot=<la dirección completa del link hacia sda>
compact default=Linux delay=40 install=menu
large-memory lba32 map=/boot/map root="<el file
system anotado>" read-only vga=normal
image=/boot/vmlinuz label=Linux
    initrd=/boot/initrd.img
image=/boot/vmlinuz.old label=LinuxOld
    initrd=/boot/initrd.img.old optional
```

En este archivo debe reemplazar `<la dirección completa del link hacia sda>` con la dirección **absoluta** hacia el *link* que anotó en el inciso c; y `<el file system anotado>` con lo que anotó en el inciso d (note que `<el file system anotado>` está rodeado de comillas).

- **¿Por qué se usa `<la dirección completa del link hacia sda>` en lugar de sólo `/dev/sda`, y cuál es el papel que el programa `udev` cumple en todo esto?**

Porque debe poder ser accedido desde cualquier directorio y es una dirección específica, como un path o echo. Udev se encarga de proporcionar eventos de dispositivo a los que estén conectados a la computadora.

- **¿Qué es un *block device* y qué significado tiene *sdxN*, donde *x* es una letra y *N* es un número, en direcciones como */dev/sdb*? Investigue y explique los conceptos de *Master Boot Record (MBR)* y *Volume Boot Record (VBR)*, y su relación con UEFI.**

Un block device es un fichero que hace dato a un mecanismo. Estos archivos se distinguen de los archivos de caracteres porque proporcionan paso al dispositivo sin demostrar las características del hardware del ingenio. El MBR es la primera conjunto de la memoria secundaria que le dice al computador cómo estomagar el sistema operante. VBR está delimitado Al principio de una partición y ayuda en el boot similarmente a cómo funciona el MBR

- **¿Qué es hacer *chain loading*?**

Es un método utilizado por programas informáticos para reemplazar el programa que se está ejecutando actualmente por un programa nuevo.

- **¿Qué se está indicando con la configuración *root="<el file system anotado>"*?**

Se especifica la sección del disco para usarse y cargar el Kernel allí.

- f. Abra, en el mismo directorio */etc*, el archivo *kernel-img.conf*, y asegúrese de que incluya las siguientes líneas (*i.e.*, modifique y agregue según sea necesario):

```
do_symlinks = yes relative_links  
= yes link_in_boot = yes
```

- g. Vaya al directorio raíz y elimine los *links* simbólicos llamados *vmlinuz* e *initrd.img*.
- h. Vaya al directorio */boot* y cree *links* simbólicos hacia *vmlinuz-3.16.0-4-686-pae* e *initrd.img-3.16.0-4-686-pae* con nombres *vmlinuz* e *initrd.img* respectivamente. Asegúrese del orden en el que se especifican los parámetros para crear un *link* simbólico (puede consultar *man ln*).

- **¿Qué es *vmlinuz*?**

Es el encargado de cargae el sistema operativo en la memoria principal. Realiza una rutina que configura el hardware al hacer el reinicio del sistema

- i. En este mismo directorio elimine el subdirectorio *grub* con el siguiente comando:


```
sudo rm -r /boot/grub
```

- j. Vaya al directorio `/etc/kernel` y ejecute `ls`. Verá varios directorios. Acceda a cada uno y elimine los archivos que encuentre (si encuentra) que tengan “grub” en su nombre.
- k. Vaya al directorio `/etc/initramfs/post-update.d` y elimine los archivos que encuentre (si encuentra) que tengan “grub” en su nombre.
- l. Ejecute el siguiente comando:

```
sudo dpkg-reconfigure linux-image-3.16.0-4-686-pae
```

- m. Si todo ha salido bien hasta ahora, reinicie su máquina virtual. Su sistema cargará el sistema operativo por medio de LILO en lugar de GRUB, y deberá iniciar sin pasar por el menú de selección de *kernel*. Cree una nueva *snapshot* de su máquina virtual y luego use esta y la *snapshot* anterior para tomar fotos del proceso de *booteo*, evidenciando el empleo de GRUB y LILO en cada caso.

Incluya sus fotos o capturas con sus entregables.

- **Mencione tres diferencias funcionales entre GRUB y LILO.**

1. GRUB se puede utilizar para varios sistemas operativos y LILO se usa solo para el sistema operativo Linux.
2. GRUB tiene una interfaz de comando interactiva y LILO no la posee.
3. Cualquier cambio realizado en `grub.conf` se utilizará automáticamente la próxima vez que se inicie el sistema. Cualquier cambio realizado en `lilo.conf` no se lee en el momento del arranque. Es necesario reinstalar el MBR.