Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Juan Luis Garcia; Juan Carlos Canteo.
Semestre I, 2022

Laboratorio #4

Fecha de Entrega: 4 de Mayo, 2022.

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 lograble 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
```

Durante la ejecución verá mucho *output*. Realice algunas acciones en su sistema operativo sin perder de vista el *output* que la terminal le muestra (*e.g.*, minimice una ventana, abra un archivo de texto, etc.)

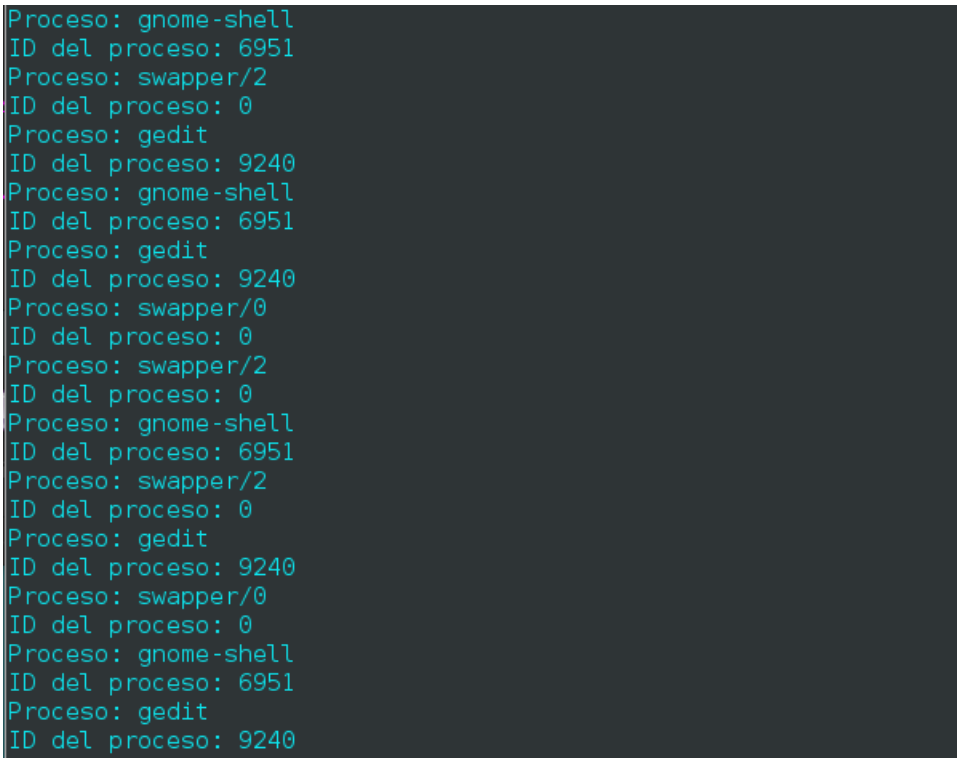


Figura 1. Ejemplo de *output* obtenido luego de ejecutar el archivo profile.stp

- ¿Qué puede ver en el *output* cuando realiza estas acciones?
  Al ejecutar el archivo creado y no realizar ninguna acción generalmente se observa el proceso ***swapper*** con id 0. Sin embargo, al momento de tener alguna interacción como abrir un archivo en un editor de texto, se cambia el proceso, en este caso a ***gedit***, como se puede ver en la **figura 1,** y de igual forma cambia el id del proceso. De igual forma, si se da *click* en la terminal, el proceso ahora es ***gnome-shell*** con un id de proceso distinto de 0.

- ¿Para qué sirve SystemTap?

  Systemtap permite monitorear las actividades de un sistema Linux en ejecución. Para hacer este monitoreo se crea un script el cual puede diseñarse para extraer información, filtrarla y resumirla de manera eficiente, permitiendo así que se puedan diagnosticar problemas de *performance* o encontrar la causa de un error o *bug*. La idea detrás de Systemtap es ser una herramienta de *tracing* que permita estudiar y monitorear las actividades del sistema operativo.
  (Red Hat, s. f.-c)

- ¿Qué es una *probe*?

  Los *scripts* de Systemtap se componen de 2 cosas principales: eventos y manejadores o *handlers*. Una *probe* por lo tal es el conjunto de un envento y su *handler*. (Red Hat, s. f.-a)

- ¿Cómo funciona SystemTap?

  Systemtap funciona mediante scripts. Como se mencionó anteriormente estos *scripts* están formados de *probes*, que no son más que un evento + su manejador. Son estos *scripts* los que indican qué tipo de información se debe de recolectar y qué se debe de hacer con esa información. (Red Hat, s. f.-a)

  El proceso, una vez que se tiene el *script* es el siguiente:
  - Se ejecuta el *script* y se inicia una *SystemTap sesión*
  - Luego, *SystemTap* traduce el *script* a C. Ejecuta el compilador de C del sistema y así crear un módulo de kernel mediante el *script*
  - *SystemTap* carga el módulo, habilitando así todas las *probes* en el *script*.
  - Se espera a que los eventos definidos en el *script* se ejecuten, y a medida que esto pasa el *handler* correspondiente se va ejecutando.
  - Una vez se finaliza la *SystemTap session* los *probes* se deshabilitan y el módulo de kernel se descarga (*unloaded)* (Red Hat, s. f.-b)

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

  Profiling se refiere al proceso de examinar, analizar, revisar y resumir información para obtener para ayudar a identificar problemas de performance o monitorear la actividad de un sistema. En este caso, como se puede observar en la **figura 1** lo que se retorna es el proceso que se está ejecutando en ese momento, debido a que se está llevando un *track* de los procesos que se están ejectuando, el tipo de *profiling* que se está haciendo es de grafo de llamadas. (colaboradores de Wikipedia, 2021)

Ejercicio 2 (30 puntos)

a. Abra su máquina virtual y tómele 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?

  Antes de establecer la diferencia entre pasar una lista vacía como parámetro en un método o usar void, es necesario comprender que en C existen dos formas de "identificar" un método, está la declaración, que como bien lo indica, declara el método, es como solo darle nombre al método y definir sus

parámetros y por otro lado está la definición, en este caso, además de hacer la declaración, se indica qué es lo que hará ese método.

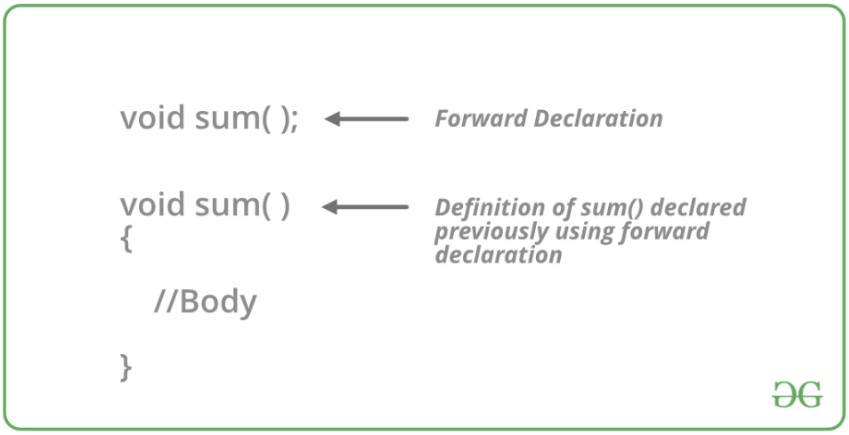


Figura 2. Diferencia entre declaración y definición de métodos en C. (GeeksforGeeks, 2019)

En una declaración de un método, al pasar como parámetro únicamente void, se le indica que el método no recibe ningún parámetro, a diferencia de si se le pasa una lista vacía de parámetros, la cual indica que se desconoce el número de parámetros que tendrá la función. (Bode, 2015)

Por otro lado, en el caso de la definición de un método utilizar void o una lista vacía de parámetros no tiene ninguna diferencia, ya que en ambos casos se indica que la función no toma ningún parámetro, sin embargo, en C nunca se debe de pasar una lista vacía de parámetros en una declaración o definición de un método (se considera de mala práctica), ya que si se espera que no tome ningún parámetro se debe de utilizar void en la lista de parámetros. (Bode, 2015)

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 Sistopspor un mensaje personalizado.

- ¿Qué diferencia hay entre printk y printf?

  Printk es un método en C de la interfaz del kernel de Linux (es por ello por lo que uno de los include en simple.c es kernel, esto para hacer uso de los métodos descritos allí). Es una de las funciones más conocidas de la interfaz ya que es una la herramienta por defecto para debuggear y dar seguimiento (*tracing*). (GeeksforGeeks, 2021)

Por otro lado, el método printf, el cual deriva su nombre de *print formatted*, muestra un mensaje en una terminal, el cual puede ser una cadena, número o cualquier formato especificado (GeeksforGeeks, 2021). Algo interesante, es que printk está basado en printf, sin embargo, en el caso de printk se puede especificar un nivel de registro o *log level*. (Kernel, s. f.)

Debido a que printk envía los mensajes al registro del kernel, esto básicamente indica que los mensajes son escritos en un *kernel log buffer*, y la forma usual de leer este buffer es mediante dmesg (Kernel, s. f.). Por otro lado, printf escribe a un archivo *std-out*. (GeeksforGeeks, 2021)

De manera general, se puede indicar que la principal diferencia es que printk es usado por el kernel para mostrar mensajes en pantalla, mientras que printf es usado por cualquier aplicación/programa. (GeeksforGeeks, 2021)

- ¿Qué es y para qué sirve KERN_INFO?

KERN_INFO es el nivel de registro (*log level)* e indica que el mensaje es para informar y tiene un nivel de prioridad de 7. El nivel de registro lo que indica es el nivel de importancia del mensaje. Con base a esto y el *console_loglevel* (la cual es una variable del kernel), el kernel determina si debe de mostrar el mensaje inmediatamente en pantalla. Por ende, si la prioridad del mensaje es más alta que el *console_loglevel* entonces el mensaje "imprimirá" en la consola. (Kernel, s. f.)

Name	String	Alias function
KERN_EMERG	"0"	`pr_emerg()`
KERN_ALERT	"1"	`pr_alert()`
KERN_CRIT	"2"	`pr_crit()`
KERN_ERR	"3"	`pr_err()`
KERN_WARNING	"4"	`pr_warn()`
KERN_NOTICE	"5"	`pr_notice()`
KERN_INFO	"6"	`pr_info()`
KERN_DEBUG	"7"	`pr_debug()` and `pr_devel()` if DEBUG is defined
KERN_DEFAULT	""	
KERN_CONT	"c"	`pr_cont()`

Figura 3. Niveles de registro disponibles. (Kernel, s. f.)

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?

  Una *goal definition* indica los archivos que se van a crear, instrucciones especiales de compilación o el acceso a subdirectorios de manera recursiva. (Kernel, s. f.-a)

  Existen dos formas de definir una *goal definition,* ***obj-***y, el cual es para *built-in object goals* y ***obj-m***, el cual se utiliza para *loadable module goals*. Por lo tanto, al hacer la definición de meta ***obj-m*** indica que los archivo objeto son creados como módulos cargables de kernel. (Kernel, s. f.-a)

  Algo interesante a mencionar es que el tipo de *Makefiles* que contiene una goal definition usualmente son *Makefiles* del kenerl de Linux.

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

  Un *Makefile* se compone de un conjunto de reglas, estás reglas, y estas reglas a su vez se componen de *targets, prerequisites* y *comands*, donde los *targets* usualmente son nombres de archivos, pero también pueden ser simplementes nombres para identificar el conjunto de comandos, este tipo de *targets* se conoce como *phony targets*. Por otro lado, los *prerequisites* o prerrequisitos son archivos que se utilizan para que el *target* sea ejecutado y finalmente los comandos, estos son el conjunto de pasos utilizados para crear el *target*. (*Makefile Tutorial by Example*, s. f.)

  Una analogía para las reglas de los *Makefile* es verlo como una función, donde el *target* es el nombre de la función, los *prerequisites* son los parámetros y los *comands* es la definición misma de la función.

Por lo mencionado anteriormente, entonces se sabe que tanto *all* como *clean* son *phony targets*, *all*, debido a que es el primer *target* en el archivo, este es el *default goal* por lo tal, es el que se ejecuta por defecto, y al ejecutar el comando make, lo que hace es ejecutar el comando que se encuentra bajo el *target all*, sin embargo, cabe resaltar que *all* es usualmente utilizado para indicar que se ejecuten múltiples *targets* por defecto. (*Makefile Tutorial by Example*, s. f.)

Por otro lado, en el caso de *clean* este es usualmente utilizado para eliminar el *output* de otros *targets*, tal como su nombre lo indica, se encarga de la limpieza de otros *targets*. (*Makefile Tutorial by Example*, s. f.)

- ¿Qué hace la opción –C en este *Makefile*?
  La opción -C después del comando *make* en el *Makefile* le indica que, antes de ejecutar el *Makefile* primero debe de cambiar al directorio "/lib/modules/$(shell uname –r)/build" y ya luego ejecutar la opción modules. (GNU, s. f.)

- ¿Qué hace la opción M en este *Makefile*?
  Algo que es importante notar es que M no es una opción sino más bien un argumento, como bien se mencionó -C realiza un cambio de directorio para buscar el *Makefile* que va a ejecutar, entonces la variable M lo que hace es que regresa al directorio del módulo antes de ejecutar los *targets* de *modules*. (4Pie0, 2013)

g. Ejecute el comando make en el directorio donde haya creado simple.cy su correspondiente
Makefile.

```
oscreader@OSC:~/Desktop/lab04/ex02$ make
make -C /lib/modules/3.16.0-4-686-pae/build M=/home/oscreader/Desktop/lab04/ex02 modules
make[1]: Entering directory '/usr/src/linux-headers-3.16.0-4-686-pae'
Makefile:10: *** mixed implicit and normal rules: deprecated syntax
make[1]: Entering directory `/usr/src/linux-headers-3.16.0-4-686-pae'
  CC [M]  /home/oscreader/Desktop/lab04/ex02/simple.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/oscreader/Desktop/lab04/ex02/simple.mod.o
  LD [M]  /home/oscreader/Desktop/lab04/ex02/simple.ko
make[1]: Leaving directory '/usr/src/linux-headers-3.16.0-4-686-pae'
oscreader@OSC:~/Desktop/lab04/ex02$
```

Figura 4. Resultado de ejecutar *make*.

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.

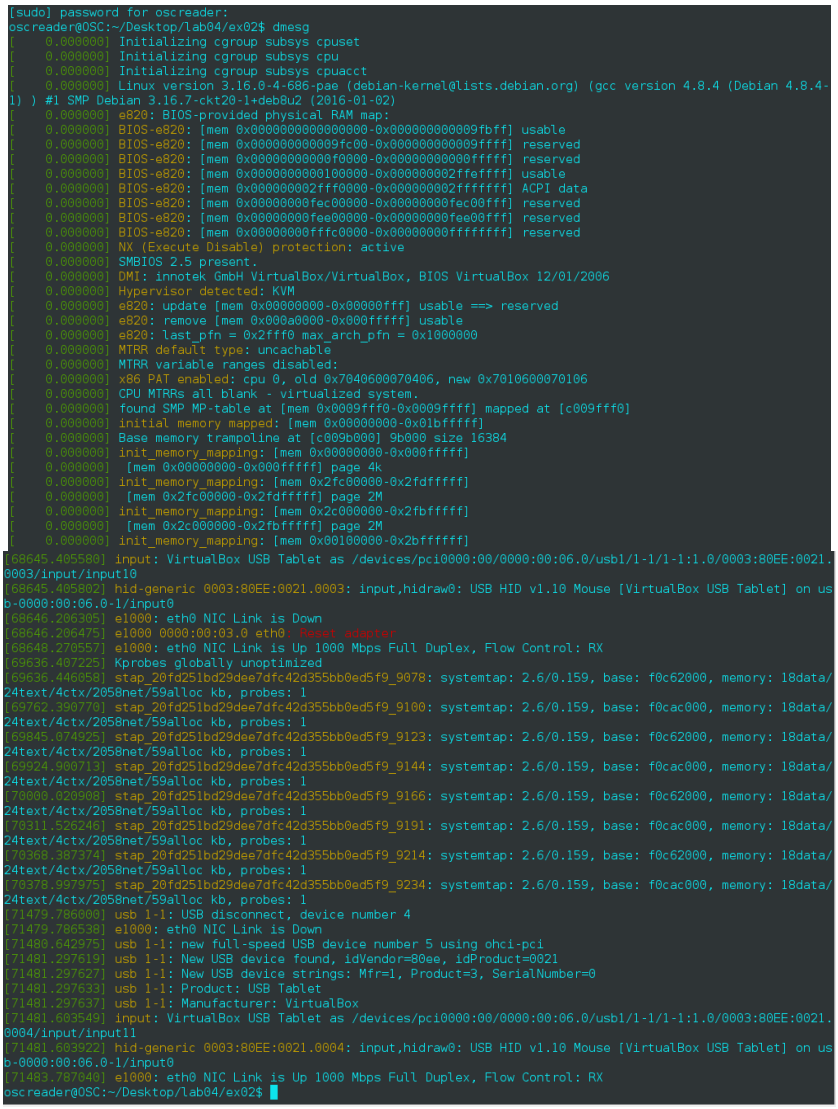


Figura 5. Resultado de ejecutar comando dmesg luego de ejecutar comando sudo insmod simple.ko.

- ¿Para qué sirve dmesg?

  Al momento de explicar acerca de la diferencia de printk y printf, se comentó un poco sobre dmesg. Básicamente dmesg es la forma de mostrar en consola lo que se ha mandado al *ring buffer* del kernel mediante printk. (McKay, 2019)

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

```
int simple_init(void){
        printk(KERN_INFO"LOADING MODULE\nOops I did it again");
        return 0;
}
```

Figura 6. Implementación de función simple_init.

Básicamente lo que hace la función simple_init es escribir en el *ring buffer* el mensaje entre comillas dentro de printk, y este mensaje lo guarda con un *log level* de KERN_INFO, el cual, como se muestra en la **figura 3** tiene una prioridad

de 6. Algo interesante es que si bien se está inicializando el módulo mediante insmod no se muestra en las capturas de la **figura 5** esto debido a que dmesg no muestra TODOS los mensajes en consola sino solo aquellos cuyo *log level* es mayor al *console_loglevel*, básicamente los que tienen alta prioridad.

i. Ahora ejecute los siguientes comandos:

```
sudo rmmod simple
dmesg
```

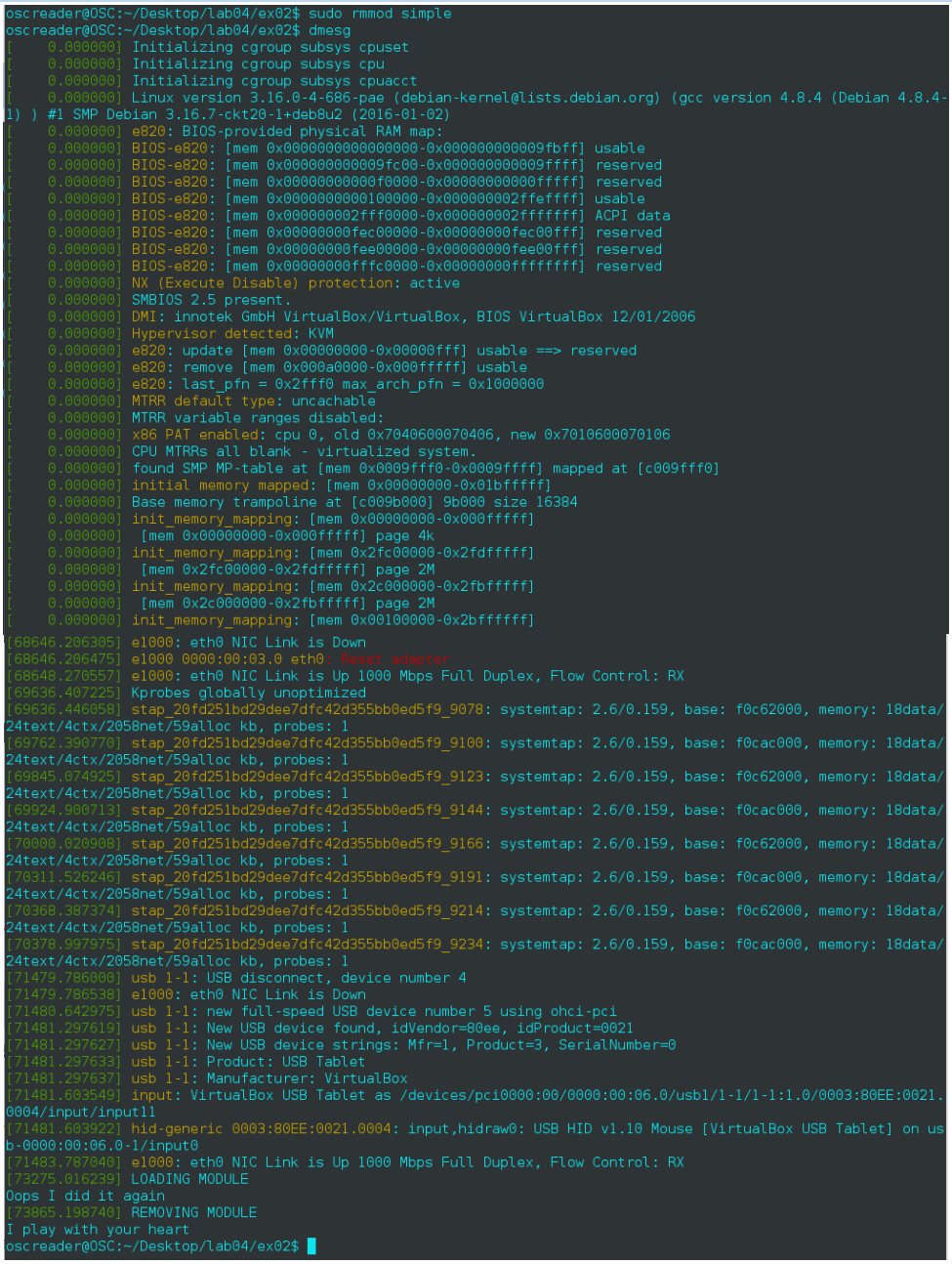


Figura 7. Resultado de ejecutar comando dmesg luego de ejecutar comando sudo rmmod simple

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

- ¿Qué hace la función simple_exiten su programa simple.c?

```
void simple_exit(void){

        printk(KERN_INFO"REMOVING MODULE\nI play with your heart");
}
```

Figura 8. Implementación de la función simple_exit.

De forma muy similar a la función simple_init, la función simple exit, básicamente solo escribe un mensaje en el *ring buffer* del kernel. En este caso el mensaje inicia con REMOVIN MODULE, ya que a diferencia de simple_init, esta función se llama al momento de eliminar el módulo. De igual forma que con el mensaje escrito por simple_init el de simple_exit cuenta con un *log_level* KERN_INFO los cuales son para mensajes de información y tienen una importancia de 6.

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

  Como bien se sabe, dentro del ordenador se tienen dos modos operando, el modo usuario y el modo kernel. Generalmente el usuario, valga la redundancia, trabaja en modo usuario, sin embargo, al ejecutar código de esta forma permite que el usuario trabaje en el modo kernel, permitiendo así tener acceso a muchas más funciones que desde el modo usuario estas están deshabilitadas, es decir da paso a mayor control y trabajo sobre el kernel en sí.

Ejercicio 3 (40 puntos)

a. Si todo ha salido bien con los demás ejercicios, tómele 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>.

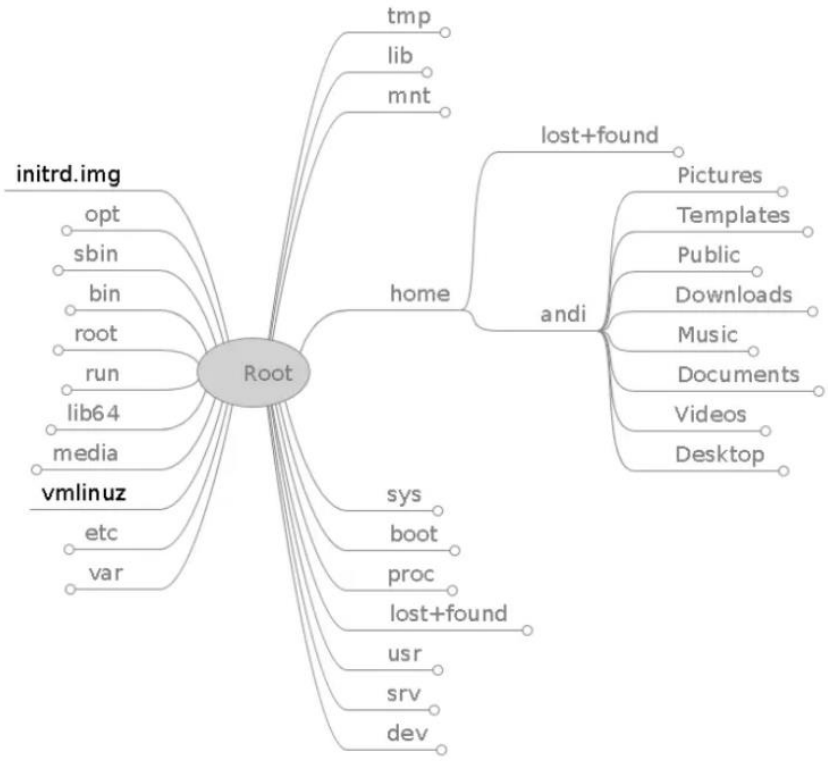


Figura 9. Jerarquía de directorios en Linux.

- ¿Qué es y para qué sirve el archivo fstab?
  Fstab hace referencia a *file system table*, es una tabla de configuraciones que se encarga de "montar/desmontar" *fyle systems* en un ordenador. Este archivo se utiliza para configurar las particiones que son necesarias que se monten desde el arranque del sistema. Se lee por la orden *mount* para poder determinar qué especificaciones utilizar al montar un dispositivo o una partición. (Gómez, s. f.)

- ¿Qué almacena el directorio /etc? ¿En Windows, quién (hasta cierto punto) funge como /etc?
  Etc es un directorio que se encarga de almacenar todos los archivos de configuración, tanto del sistema operativo, como de aplicaciones que se instalen en el ordenador. Algo que se debe de tomar en cuenta es que este directorio no debe de contener binarios, ya que estos se guardan en /sbin o /bin. (*Linux: El Directorio «/Etc» - [Admin101]*, 2020)

  En Windows, el que se podía considerar de cierta manera como un equivalente a /etc es el Windows *registry*, ya que como bien se sabe, el *registry* se encarga de almacenar la información de configuración del sistema operativo para uno o más usuarios, aplicaciones, dispositivos de hardware. (Han, 2022)

- ¿Qué se almacena en /dev ý en /dev/disk?
  Como bien se sabe, en Linux, todo es manejado como un archivo, por lo tal en el directorio /dev se encuentran todos los dispositivos de almacenamiento conectados al sistema, representados como un archivo. (*Linux: El Directorio*

"/Dev" - [Admin101], 2020)

Por lo mencionado anteriormente, entonces es fácil pensar que el directorio /dev/disk almacena información del disco del sistema, y efectivamente, en /dev/disk se encuentra la información de todos los discos que se encuentran conectados al sistema, además se encarga de su gestión y organización. (*/Dev/Disk*, s. f.)

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).

a. ¿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?

Como bien se sabe una ruta relativa es aquella que se forma a partir del directorio actual, por otro lado, la ruta absoluta indica de manera explícita la ruta de acceso iniciando desde el directorio raíz. En este caso se utiliza la ruta absoluta ya que así, es posible acceder al enlace correctamente desde cualquier directorio evitando errores de relatividad.

Como manera curiosa, al inicio, cuando estaba elaborando esta parte del laboratorio no había colocado la ruta absoluta, sino únicamente la ruta relativa, que era la que se encontraba en el archivo, y al momento de probar se generaba un error indicando que no podía encontrar el enlace especificado.

Udev en Linux procede de *userspace /dev* y básicamente se encarga de la detección y gestión de los dispositivos, entre sus funciones se encuentra la creación de enlaces simbólicos en el directorio /dev para poder acceder a los dispositivos, lo cual es justamente lo que se hace en este apartado, se utiliza un enlace simbólico para acceder a un dispositivo/partición, y este enlace es creado por udev. (Orovengua, 2020)

b. ¿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 Rercord* (VBR), y su relación con UEFI.

Un *block device* es un archivo que hace referencia a un dispositivo, este archivo da acceso al control del dispositivo abstrayendo del usuario todas las características del hardware (Homer, 2016). Por otro lado, sdxN hace referencia a nombres de los discos o dispositivos de almacenamiento, estos nombres se encuentran ordenados de manera alfabética, por ello x es una letra, representando la a que es el primer disco o el disco principal. Por otro lado, el número hace referencia al número de partición. (A, 2021)

El *Master Boot Record* es la información de cualquier sector del disco que indica en dónde y cómo se encuentra el sistema operativo para cargarlo a la computadora. También se le suele conocer como *partition table* ya que incluye una tabla en la cual almacena las particiones en las cuales se ha configurado el disco duro. (Contributor, 2005)

Volumen boot record también conocido como *partition boot sector* se encuentra en una partición particular de un dispositivo de almacenamiento y generalmente contiene el código necesario para iniciar el proceso de carga o *boot*. El programa que se utiliza para cargar el sistema operativo se conoce como *volumen boot code* y es uno de los componentes de *VBR*. (Fisher, 2021)

UEFI es un firmware, que entre otras cosas se encarga de iniciar, configurar y hacer test a todo el equipo, validando que todo el hardware está en buen estado. Además, mediante la combinación con MBR y VBR permite la carga del sistema operativo al ordenador. Algo que es necesario recalcar es que no es muy usual que UEFI se utilice con MBR sino más bien con GPT, sin embargo, debido a que estos pueden convertirse mutuamente, también es posible utilizar el MBR con UEFI. (Manjaro, 2021)

c. ¿Qué es hacer *chain loading*?

Al cargar sistemas operativos, esto se puede hacer de forma directa o indirecta, al hacerlo de forma indirecta es donde entra el concepto de *chain loading* y esto generalmente lo hacen los sistemas que no soportan *multiboot*. *Chain loading* entonces es el proceso en el cual un *boot loader* carga otro *boot loader* para iniciar así el proceso de carga (Mahood, 2016), este proceso se muestra en

el siguiente diagrama:

Figura 10. Proceso de *chain loading*.

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

Con esta configuración se está definiendo el directorio *root* o raíz y este se define com el *file system* que se anotó anteriormente, que apunta al directorio raíz conocido en Linux "/". Recordemos que el directorio raíz es el nodo principal en la jerarquía de directorios de Linux, ya que en este se contienen todos los archivos y directorios necesarios para el correcto funcionamiento del sistema.

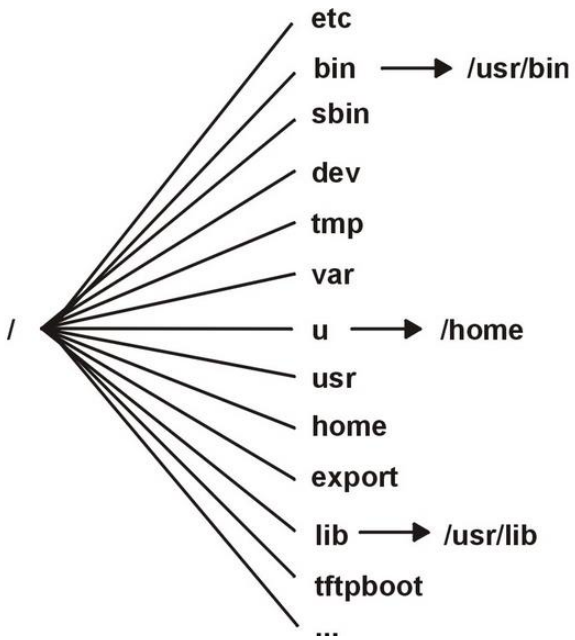


Figura 11. Subdirectorios del *root file system*. (IBM, s. f.)

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 vmlinuze 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).
    a. ¿Qué es vmlinuz?

        Vmlinuz básicamente es el ejecutable del kernel de Linux. Este puede ser el ejecutable en sí o un enlace directo al ejecutable. Vmlinuz es el kernel de manera comprimida y *booteable*, es decir capaz es capaz de cargar el sistema operativo a memoria, permitiendo así que el ordenador sea funcional para el usuario. (Belleuve Linux, s. f.)

i. En este mismo directorio elimine el subdirectorio grubcon 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.

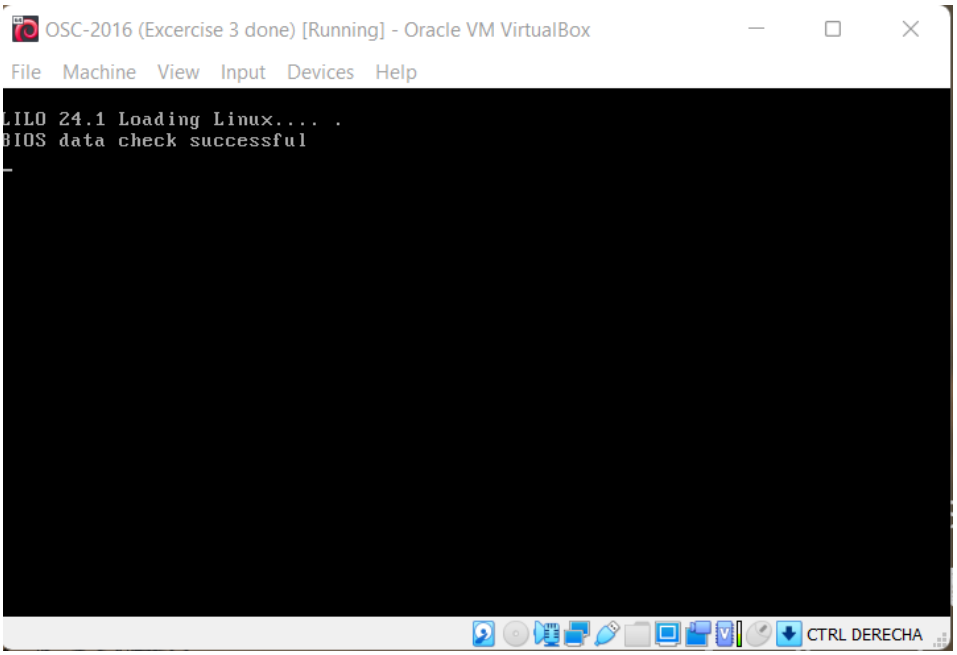
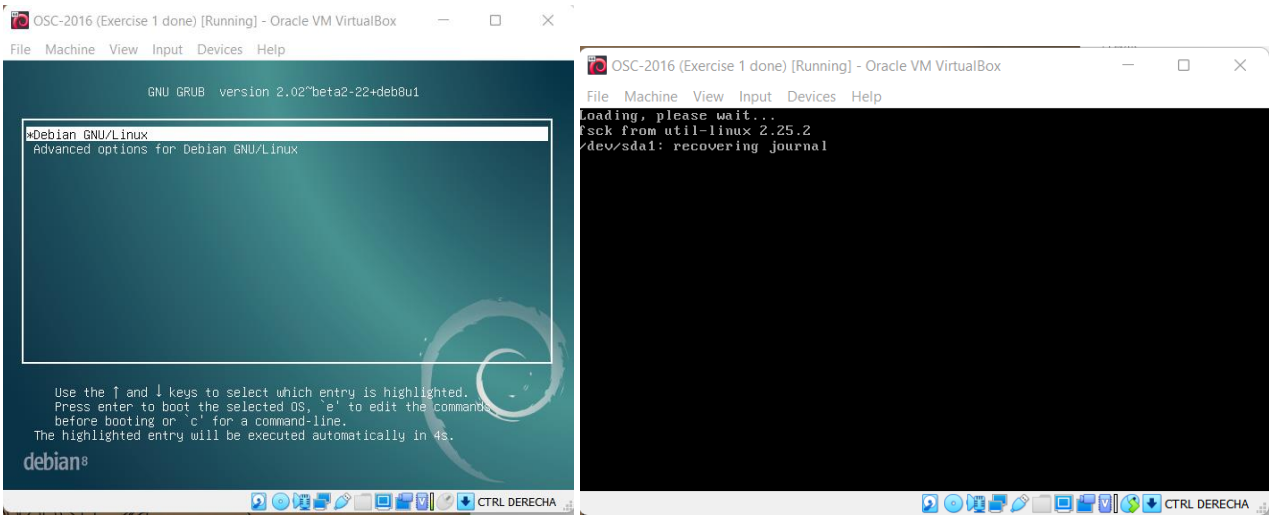


Figura 12. Evidencia de proceso de *booteo* con LILO



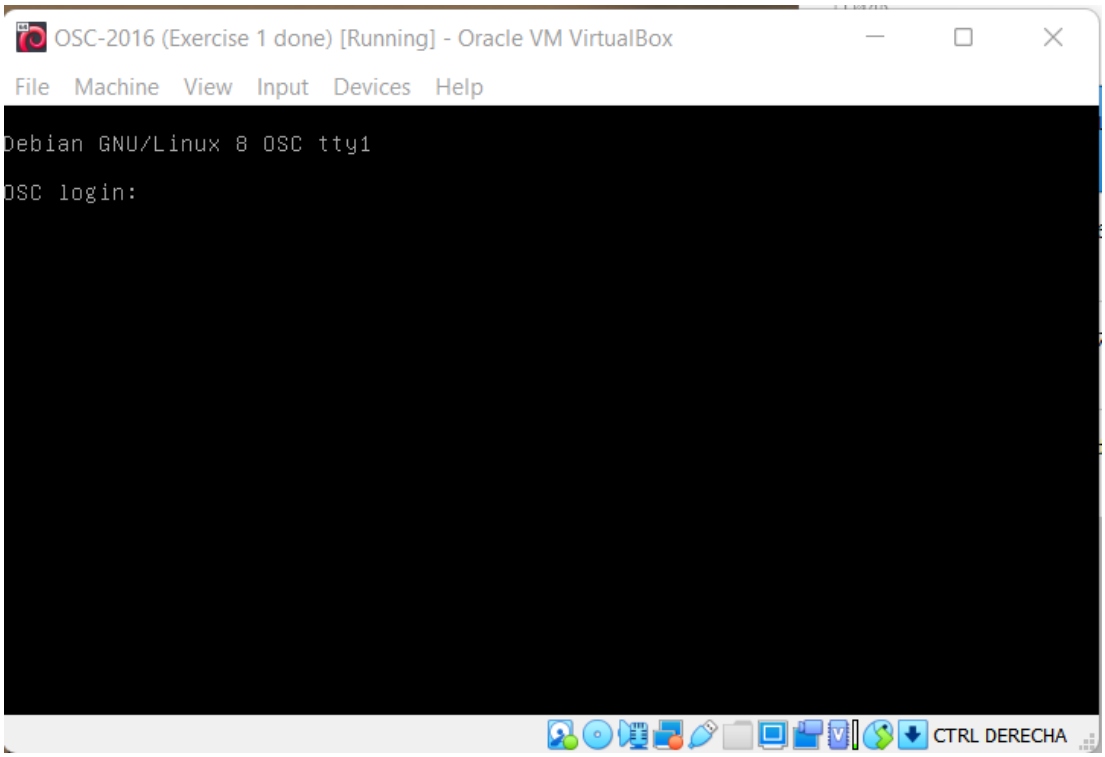


Figura 13. Evidencia de proceso de *booteo* mediante GRUB

a. Mencione tres diferencias funcionales entre GRUB y LILO.

GRUB	LILO
Solo la primera etapa de GRUB se almacena en el MBR	Se almacena en el sector de arranque del disco duro MBR. Esto implica que si se hacen modificaciones a la configuración de LILO o se instala un nuevo kernel se debe de instalar nuevamente LILO en el MBR. Z(LILO, s. f.)
Los cambios en la configuración se leen automáticamente durante el proceso de arranque lo que permite que si existe un error se arregle en ese momento	Al realizar un cambio en el gestor de arranque, se debe de reescribir el MBR. Lo que puede conllevar a errores y problemas de encendido
Permite el método de encriptación de contraseñas MD5	Solo admite contraseñas de texto
Permite la lectura de particiones de tipo ext2	No puede leer particiones de tipo ext2

Bibliografía

4Pie0. (2013, 30 noviembre). *M option in make command, Makefile*. Stack Overflow. https://stackoverflow.com/questions/20301591/m-option-in-make-command-makefile

A. (2021, 25 octubre). *¿Qué es SDA y SDB en Linux?* CompuHoy.com. https://www.compuhoy.com/que-es-sda-y-sdb-en-linux/

Belleuve Linux. (s. f.). *vmlinuz Definition*. linfo.org. http://www.linfo.org/vmlinuz.html

Bode, J. (2015, 11 junio). *What is the difference between function() and function(void)?* Software Engineering Stack Exchange. https://softwareengineering.stackexchange.com/questions/286490/what-is-the-difference-between-function-and-functionvoid

Chainloading in Linux - LinuxQuestions.org. (s. f.). Linuxquestions. https://www.linuxquestions.org/linux/answers/Applications_GUI_Multimedia/Chainloading_in_Linux

colaboradores de Wikipedia. (2021, 5 agosto). *Análisis de rendimiento de software*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/An%C3%A1lisis_de_rendimiento_de_software#Tipos_de_profilers_basados_en_su_salida

Contributor, T. (2005, 5 abril). *Master Boot Record (MBR)*. WhatIs.Com. https://www.techtarget.com/whatis/definition/Master-Boot-Record-MBR

/dev/disk. (s. f.). Tcl. https://wiki.tcl-lang.org/page/%2Fdev%2Fdisk

Fisher, T. (2021, 10 julio). *What Is a Volume Boot Record?* Lifewire. https://www.lifewire.com/volume-boot-record-2625818

GeeksforGeeks. (2019, 28 noviembre). *What are Forward declarations in C++*. https://www.geeksforgeeks.org/what-are-forward-declarations-in-c/

GeeksforGeeks. (2021, 2 julio). *Difference between Printk() and Printf() in Linux*. https://www.geeksforgeeks.org/difference-between-printk-and-printf-in-linux/

GNU. (s. f.). *Options Summary (GNU make)*. gnu.org. https://www.gnu.org/software/make/manual/html_node/Options-Summary.html

Han, D. (2022, 6 abril). *Windows registry for advanced users - Windows Server*. Microsoft Docs. https://docs.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users

Homer, M. (2016, 1 febrero). *What is a block device?* Unix & Linux Stack Exchange. https://unix.stackexchange.com/questions/259193/what-is-a-block-device

IBM. (s. f.). *Root file system*. Ibm.Com. https://www.ibm.com/docs/pl/aix/7.1?topic=tree-root-file-system

Kernel. (s. f.-a). *Linux Kernel Makefiles*. kernel.org. https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt

Kernel. (s. f.-b). *Message logging with printk — The Linux Kernel documentation*. Kernel.Org. https://www.kernel.org/doc/html/latest/core-api/printk-basics.html

LILO. (s. f.). *LILO*. web.mit.edu. http://web.mit.edu/rhel-doc/3/rhel-rg-es-3/s1-grub-lilo.html

Linux: El directorio "/dev" - [admin101]. (2020, 7 mayo). Medium. https://medium.com/@admin101/linux-el-directorio-dev-parte-4-5cdb140ac28

Linux: El directorio «/etc» - [admin101]. (2020, 7 mayo). Medium. https://medium.com/@admin101/linux-el-directorio-etc-e792141da30e

Mahood, T. (2016, 14 abril). *Explanation of chain loading in boot loaders?* Super User. https://superuser.com/questions/1065392/explanation-of-chain-loading-in-boot-loaders

Makefile Tutorial by Example. (s. f.). Makefile Tutorial. https://makefiletutorial.com/

Manjaro. (2021, 7 septiembre). *Some basics of MBR v/s GPT and BIOS v/s UEFI*. https://wiki.manjaro.org/index.php/Some_basics_of_MBR_v/s_GPT_and_BIOS_v/s_UEFI

Marquez, V. (s. f.). *Directorios*. dis.um.es. http://dis.um.es/%7Elopezquesada/documentos/IES_1213/LMSGI/curso/xhtml/xhtml11/Paginas/Directorios.html

McKay, D. (2019, 5 diciembre). *How to Use the dmesg Command on Linux*. How-To Geek. https://www.howtogeek.com/449335/how-to-use-the-dmesg-command-on-linux/

Orovengua, J. (2020, 10 junio). *Cómo usar Udev para la detección y gestión de dispositivos en Linux*. LinuxParty. https://www.linuxparty.es/75-hardware/10488-como-usar-udev-para-la-deteccion-y-gestion-de-dispositivos-en-linux.html

Red Hat. (s. f.-a). *3.2. SystemTap Scripts Red Hat Enterprise Linux 7*. Red Hat Customer Portal. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/systemtap_beginners_guide/scripts

Red Hat. (s. f.-b). *Chapter 3. Understanding How SystemTap Works Red Hat Enterprise Linux 5*. Red Hat Customer Portal. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/systemtap_beginners_guide/understanding-how-systemtap-works

Red Hat. (s. f.-c). *SystemTap Beginners Guide Red Hat Enterprise Linux 7*. Red Hat Customer Portal. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/systemtap_beginners_guide/index