

Universidad del Valle De Guatemala

Facultad de Ingeniería

Sistemas Operativos



Lab 4

Javier Mombiola

Carne: 20067

Sección: 21

Guatemala, 23 de marzo 2023

Ejercicio 1

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.)



```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$ sudo stap profiler.stp
Proceso: swapper/0
ID del proceso: 0
Proceso: swapper/1
ID del proceso: 0
Proceso: swapper/2
ID del proceso: 0
Proceso: swapper/3
ID del proceso: 0
Proceso: swapper/2
ID del proceso: 0
Proceso: swapper/0
ID del proceso: 0
Proceso: swapper/3
ID del proceso: 0
Proceso: swapper/1
```

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

- A la hora de realizar las diferentes acciones como: minimizar ventanas, abrir otros programas, etc., mientras el programa profiler.stp esta corriendo, podemos ver que se muestra la información sobre los procesos que se están ejecutando en ese momento, incluyendo el nombre del proceso y su ID.

¿Para qué sirve SystemTap?

- SystemTap es una herramienta de diagnóstico y monitoreo de Linux. Esta herramienta sirve para poder recopilar información sobre el rendimiento y el comportamiento del sistema, y para realizar análisis y solucionar problemas en tiempo real.

¿Qué es una probe?

- Una probe es una sección de código que se inserta en el kernel de Linux para recopilar información específica. Las probes se utilizan en SystemTap para recopilar datos sobre el rendimiento y el comportamiento del sistema.

¿Cómo funciona SystemTap?

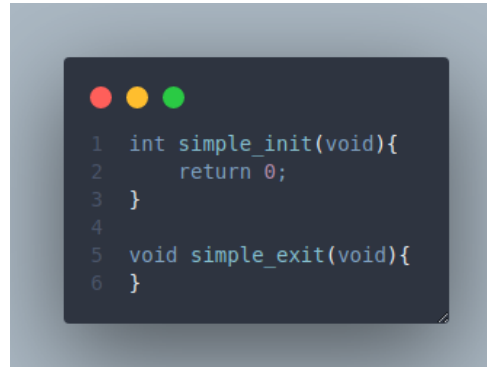
- Esta herramienta funciona mediante la inserción de código en el kernel de Linux en tiempo de ejecución. Este código se ejecuta en el contexto del sistema operativo y tiene acceso a información detallada sobre el rendimiento y el comportamiento del sistema.

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

- Profiling es el proceso de recopilar y analizar datos sobre el rendimiento y el comportamiento de un sistema o aplicación, con el objetivo de identificar cuellos de botella, ineficiencias o áreas de mejora. En este ejercicio se está haciendo profiling de los procesos que se están ejecutando en el sistema operativo en tiempo real. Este tipo de profiling se conoce como profiling dinámico.

Ejercicio 2

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.



```
1 int simple_init(void){
2     return 0;
3 }
4
5 void simple_exit(void){
6 }
```

¿Cuál es la diferencia en C entre un método que no recibe parámetros y uno que recibe `void`?

- En C la diferencia entre un método que no recibe parámetros y uno que recibe `void`, es que el método que no recibe parámetros no especifica ningún tipo de parámetros en su declaración, lo que significa que el método puede aceptar cualquier número y tipo de argumentos. Por otro lado, el método que recibe `void`, especifica que no espera ningún argumento en su declaración, lo que significa que si se intenta llamar a la función con algún argumento, este producirá un error.

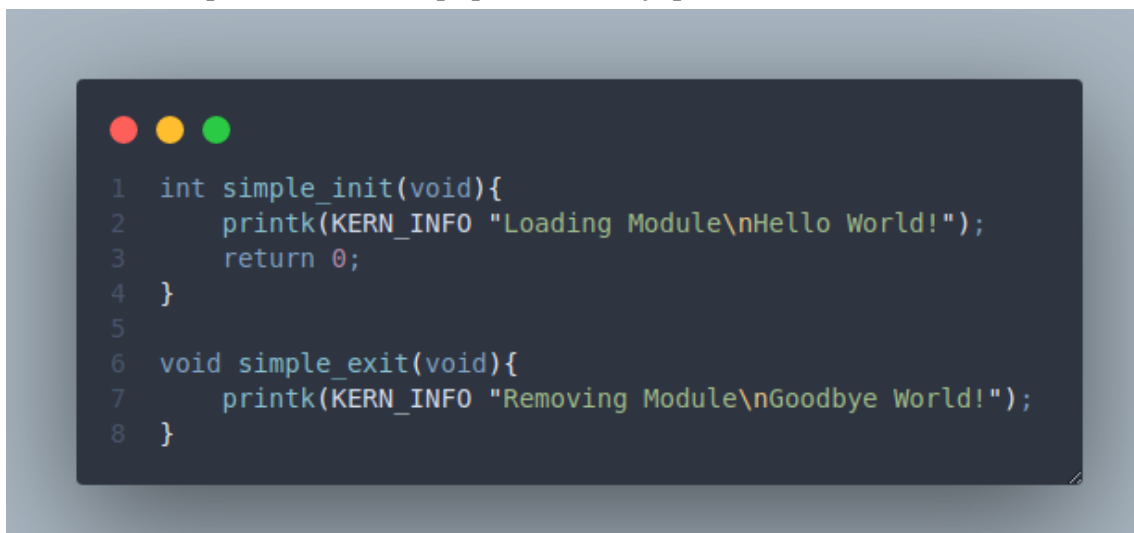
En el primer método incluya la siguiente instrucción:

```
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, reemplacé el texto `Sistops` por un mensaje personalizado.



```
1 int simple_init(void){
2     printk(KERN_INFO "Loading Module\nHello World!");
3     return 0;
4 }
5
6 void simple_exit(void){
7     printk(KERN_INFO "Removing Module\nGoodbye World!");
8 }
```

¿Qué diferencia hay entre printk y printf?

- La diferencia entre estas dos funciones es que printk() es una función a nivel de kernel que puede imprimir en diferentes niveles de registro definidos en <linux/kernel.h>, mientras que printf() siempre imprimirá en un descriptor de archivo - STD_OUT. Entonces la principal diferencia es la capacidad de printk() para especificar un nivel de registro.

¿Qué es y para qué sirve KERN_INFO?

- KERN_INFO es una macro que se utiliza en el kernel de Linux para indicar el nivel de severidad de un mensaje que se está imprimiendo. Este se utiliza, junto con printk, para poder imprimir mensajes en el registro del kernel. El nivel de registro especifica la importancia de un mensaje. El kernel decide si mostrar el mensaje inmediatamente, dependiendo de su nivel de registro y el valor actual de console_loglevel.

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>");
```

```
1 //Javier Mombiela
2 //Carnet: 20067
3 //Seccion: 21
4
5 #include <linux/init.h>
6 #include <linux/module.h>
7 #include <linux/kernel.h>
8 #include <linux/list.h>
9
10 int simple_init(void){
11     printk(KERN_INFO "Loading Module\nHello World!");
12     return 0;
13 }
14
15 void simple_exit(void){
16     printk(KERN_INFO "Removing Module\nGoodbye World!");
17 }
18
19 module_init(simple_init);
20 module_exit(simple_exit);
21 MODULE_LICENSE("GPL");
22 MODULE_DESCRIPTION("No se me ocurrió nada bueno.");
23 MODULE_AUTHOR("Javier")
```

Cree un archivo Makefile para su programa

```
1 obj-m += simple.o
2 all:
3     make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
4 clean:
5     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 definición de meta o definición de objetivo en un Makefile especifica un objetivo a construir y sus dependencias. En este Makefile, obj-m es una variable que especifica los archivos objeto que se deben construir como módulos del kernel cargables.

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

- Las líneas all y clean son objetivo en el Makefile. El objetivo all especifica la acción predeterminada que se tomara cuando se ejecute el comando make sin ningún argumento. En este caso, construye el módulo del kernel invocando make en el sistema de compilación del kernel con los argumentos apropiados.
- El objetivo clean especifica la acción que se debe tomar para limpiar los artefactos de compilación.

¿Qué hace la opción -C en este Makefile?

- La opción -C en este Makefile especifica el directorio donde make debe buscar el Makefile y realizar sus acciones. En este caso, cambia el directorio del sistema y de compilación del kernel para la versión del kernel que se esta ejecutando actualmente.

¿Qué hace la opción M en este Makefile?

- La opción M en este Makefile especifica el directorio que contiene el código fuente del módulo del kernel externo. En este caso, esta configurado en el directorio de trabajo actual. Esto permite que el Makefile pueda encontrar el archivo simple.c y producir el archivo simple.o en el mismo directorio.

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

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$
make
make -C /lib/modules/5.15.0-56-generic/build M=/home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4 modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-56-generic'
  CC [M] /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/simple.o
  MODPOST /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/Module.symvers
  CC [M] /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/simple.mod.o
  LD [M] /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/simple.ko
  BTF [M] /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/simple.ko
Skipping BTF generation for /home/javimombiela/Documents/GitHub/SistemasOperativos/Lab4/simple.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-56-generic'
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$
```

Ejecute los siguientes comandos:

```
sudo insmod simple.ko
```

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$  
sudo insmod simple.ko  
[sudo] password for javimombiela:  
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$
```

```
dmesg
```

```
[ 1490.777335] hid-generic 0003:80EE:0021.0002: input,hidraw0: USB HID v1.10 Mou  
se [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0  
[ 5889.266504] Loading Module  
Hello World!  
[ 6884.905083] Removing Module  
Goodbye World!  
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab4$
```

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

- Dicha función se llama cuando el módulo del kernel se descarga del kernel. En este caso, se imprime un mensaje en el registro del kernel utilizando la función `printk` para indicar que el módulo está siendo eliminado.

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

- Cargar y descargar módulos del kernel nos permite agregar o eliminar dinámicamente funcionalidad del kernel de Linux sin tener que reconstruir o reiniciar el sistema. Esto puede ser muy poderoso ya que nos permite ejecutar código en el espacio del kernel con acceso completo a todos los recursos y hardware del sistema.

Ejercicio 3

Vaya al directorio `/dev/disk/by-id` y ejecute el comando `ls -Al`. El resultado le mostrará varios links simbólicos (si está utilizando la máquina virtual de Linux Mint los links están marcados en un celeste brillante), 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`. Además, anote la ruta completa de la ubicación del link.

```
javimombiela@javimombiela-VirtualBox:/dev/disk/by-id$ ls -Al
total 0
lrwxrwxrwx 1 root root 9 Mar 23 10:25 ata-VBOX_CD-ROM_VB2-01700376 -> ../../sr0
lrwxrwxrwx 1 root root 9 Mar 23 10:25 ata-VBOX_HARDDISK_VBcd24ba89-61a4c117 ->
../../sda
lrwxrwxrwx 1 root root 10 Mar 23 10:25 ata-VBOX_HARDDISK_VBcd24ba89-61a4c117-par
t1 -> ../../sda1
lrwxrwxrwx 1 root root 10 Mar 23 10:25 ata-VBOX_HARDDISK_VBcd24ba89-61a4c117-par
t2 -> ../../sda2
lrwxrwxrwx 1 root root 10 Mar 23 10:25 ata-VBOX_HARDDISK_VBcd24ba89-61a4c117-par
t3 -> ../../sda3
javimombiela@javimombiela-VirtualBox:/dev/disk/by-id$
```

Link: `ata-VBOX_HARDDISK_VBcd24ba89-61a4c117`

Ruta completa: `/dev/disk/by-id/ata-VBOX_HARDDISK_VBcd24ba89-61a4c117`

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>.

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda3 during installation
UUID=80000e67-887b-44cc-9d5e-38c63c059012 / ext4 errors=remoun
# /boot/efi was on /dev/sda2 during installation
UUID=3250-D255 /boot/efi vfat umask=0077 0 1
/swapfile none swap sw
```

Contenido de la columna deseada: `UUID=80000e67-887b-44cc-9d5e-38c63c059012`

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

- El archivo `fstab` es un archivo de configuración en los sistemas basados en Unix que se encuentra en el directorio `/etc`. Este archivo contiene información estática sobre los sistemas de archivos que el sistema debe montar al iniciarse. `fstab` especifica cómo se deben montar las particiones de disco y otros dispositivos de almacenamiento en el sistema.

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

- Dicho directorio almacena archivos de configuración del sistema y scripts de inicio. Estos archivos son utilizados por el sistema y las aplicaciones para configurar su comportamiento.
- En windows, el directorio que cumple una función similar a la de /etc es el directorio C:\Windows\System32.

¿Qué se almacena en /dev y en /dev/disk?

- El directorio /dev contiene archivos especiales que representan dispositivos en el sistema. Estos archivos permiten a las aplicaciones interactuar con los dispositivos como si fueran archivos regulares.
- El subdirectorio /dev/disk contiene enlaces simbólicos a dispositivos de almacenamiento como discos duros y unidades de USB.

En ese mismo directorio /etc cree un archivo llamado lilo.conf

```
GNU nano 6.2 lilo.conf *
boot=/dev/disk/by-id/ata-VBOX_HARDDISK_VBcd24ba89-61a4c117
compact
default=Linux
delay=40
install=menu
large-memory
lba32
map=/boot/map
root="UUID=80000e67-887b-44cc-9d5e-38c63c059012"
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

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

¿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?

- Se utiliza la dirección completa del enlace hacia sda en lugar de simplemente /dev/sda para asegurarse de que el dispositivo correcto sea utilizado, incluso si el orden de los dispositivos cambia.
- El programa udev es responsable de administrar los archivos de dispositivos en /dev y crear enlaces simbólicos en /dev/disk/by-id que apuntan a los dispositivos utilizando identificadores únicos. Esto permite referenciar dispositivos de manera más robusta utilizando estos identificadores únicos en lugar de nombres de dispositivos que pueden cambiar.

¿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 tipo de dispositivo que almacena datos en bloques y permite el acceso aleatorio a estos bloques. Los discos duros y las unidades USB son ejemplos comunes de block devices. En direcciones como /dev/sdb, sd indica que se trata de un dispositivo SCSI o SATA, x es una letra que identifica el dispositivo y N es un número que identifica la partición dentro del dispositivo.
- El Master Boot Record (MBR) es el primer sector de un dispositivo de almacenamiento y contiene información sobre cómo están organizadas las particiones en el dispositivo y código para iniciar el proceso de arranque. El Volume Boot Record (VBR) es el primer sector de una partición y contiene código para cargar el sistema operativo instalado en esa partición. UEFI es una interfaz entre el firmware del sistema y el sistema operativo que reemplaza al BIOS tradicional. UEFI puede utilizar una tabla de particiones diferente llamada GPT en lugar del MBR.

¿Qué es hacer chain loading?

- Hacer chain loading significa cargar un cargador de arranque desde otro cargador de arranque. Esto permite encadenar varios cargadores de arranque para permitir la selección entre múltiples sistemas operativos o versiones del kernel.

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

- Con la configuración root="<el file system anotado>", se está indicando cuál es el sistema de archivos raíz que debe ser montado durante el proceso de arranque. Esto le dice al kernel dónde encontrar los archivos necesarios para continuar con el proceso de arranque.

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):

```
GNU nano 6.2 kernel-img.conf *
# Kernel Image management overrides
# See kernel-img.conf(5) for details
do_symlinks = yes
relative_links = yes
link_in_boot = yes
do_bootloader = no
█
```

Vaya al directorio /boot y elimine los links simbólicos llamados vmlinuz e initrd.img.

```
sudo rm vmlinuz initrd.img
javitombiela@javitombiela-VirtualBox:/boot$ sudo rm vmlinuz initrd.img
javitombiela@javitombiela-VirtualBox:/boot$ █
```

En el directorio /boot cree links simbólicos hacia vmlinuz-<su versión de kernel> e initrd.img-<su versión de kernel> con nombres vmlinuz e initrd.img respectivamente.

```
sudo ln -s vmlinuz-5.15.0-56-generic vmlinuz
sudo ln -s initrd.img-5.15.0-56-generic initrd.img
```

```
javimombiela@javimombiela-VirtualBox:/boot$ sudo ln -s vmlinuz-5.15.0-56-generic vmlinuz
javimombiela@javimombiela-VirtualBox:/boot$ sudo ln -s initrd.img-5.15.0-56-generic initrd.img
javimombiela@javimombiela-VirtualBox:/boot$
```

¿Qué es vmlinuz?

- El archivo "vmlinuz" es el núcleo del sistema operativo Linux. Es un archivo binario ejecutable que contiene el kernel de Linux comprimido, y es utilizado por el sistema operativo durante el proceso de arranque (boot) para cargar el kernel en memoria y luego iniciarlo. Se usa típicamente junto con un cargador de arranque, como GRUB o LILO, que carga el kernel en la memoria y lo inicia.

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.

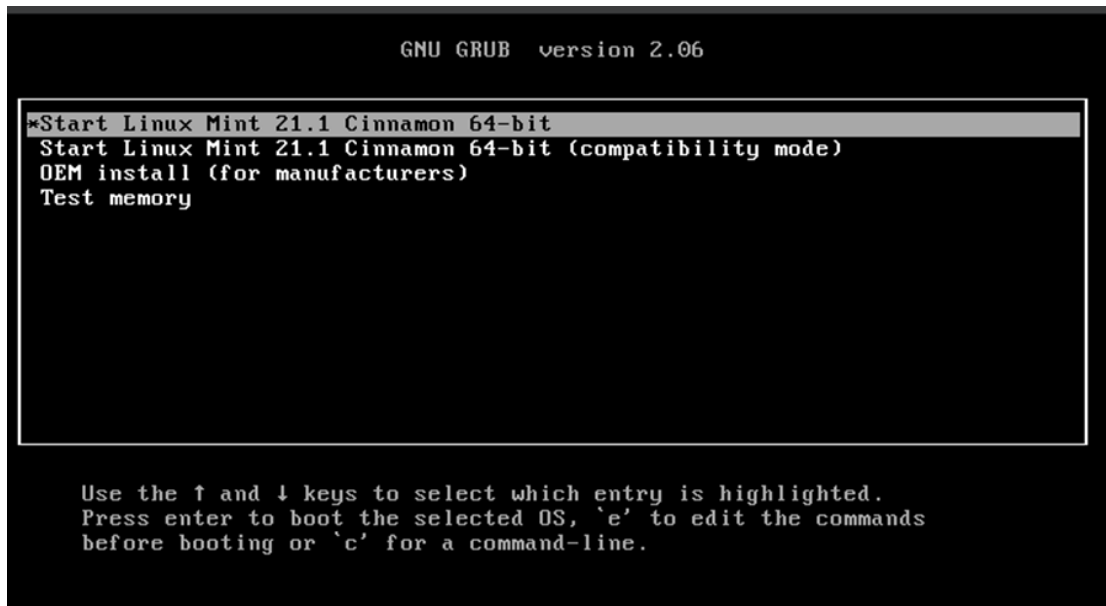
```
javimombiela@javimombiela-VirtualBox:/etc/kernel/postinst.d$ sudo rm zz-update-grub
javimombiela@javimombiela-VirtualBox:/etc/kernel/postinst.d$ ls
dkms  initramfs-tools  pm-utils  vboxadd  xx-update-initrd-links  zz-runlilo
javimombiela@javimombiela-VirtualBox:/etc/kernel/postinst.d$ cd ..
javimombiela@javimombiela-VirtualBox:/etc/kernel$ ls
header_postinst.d  install.d  postinst.d  postrm.d  preinst.d  prerm.d
javimombiela@javimombiela-VirtualBox:/etc/kernel$ cd postrm.d/
javimombiela@javimombiela-VirtualBox:/etc/kernel/postrm.d$ ls
initramfs-tools  zz-runlilo  zz-update-grub
javimombiela@javimombiela-VirtualBox:/etc/kernel/postrm.d$ sudo rm zz-update-grub
b
```

Ejecute el siguiente comando:

```
sudo dpkg-reconfigure linux-image-<su versión de kernel>
```

```
javimombiela@javimombiela-VirtualBox:~$ sudo dpkg-reconfigure linux-image-5.15.0-56-generic
/etc/kernel-img.conf:4: W: ignoring unknown parameter relative_links
I: /boot/vmlinuz.old is now a symlink to vmlinuz-5.15.0-56-generic
Processing triggers for linux-image-5.15.0-56-generic (5.15.0-56.62) ...
/etc/kernel/postinst.d/dkms:
 * dkms: running auto installation service for kernel 5.15.0-56-generic
...done.
/etc/kernel/postinst.d/initramfs-tools:
update-initramfs: Generating /boot/initrd.img-5.15.0-56-generic
/etc/kernel/postinst.d/vboxadd:
VirtualBox Guest Additions: Building the modules for kernel 5.15.0-56-generic.
update-initramfs: Generating /boot/initrd.img-5.15.0-56-generic
/etc/kernel/postinst.d/zz-runlilo:
Added Linux *
Added LinuxOld
javimombiela@javimombiela-VirtualBox:~$
```

Screenshots de proceso de booteo de GRUB



Screenshots de proceso de booteo de LILO



Mencione tres diferencias funcionales entre GRUB y LILO

1. Interfaz de comandos interactiva: GRUB tiene una interfaz de comandos interactiva, mientras que LILO no la tiene.
2. Soporte de arranque desde una red: GRUB admite el arranque desde una red, mientras que LILO no lo hace.
3. Almacenamiento de información: LILO almacena información sobre la ubicación de los sistemas operativos que puede cargar físicamente en el MBR (Master Boot Record). Si cambias tu archivo de configuración de LILO, debes reescribir el cargador de arranque de la primera etapa de LILO en el MBR. En comparación con GRUB, esta es una opción mucho más arriesgada ya que un MBR mal configurado podría dejar el sistema sin capacidad de arranque. Con GRUB, si el archivo de configuración está configurado incorrectamente, simplemente se usará la interfaz de línea de comandos de GRUB por defecto.