



Universidad  
Nacional  
de Córdoba



Facultad de  
Ciencias Exactas  
Físicas y Naturales

# Trabajo Práctico #4

## *“Módulos del Kernel”*

**Asignatura:** Sistemas de computación

APELLIDO Y NOMBRE	CARRERA	MATRICULA
Corvalán, Abel Nicolás	Ingeniería Electrónica	41.220.050
Soria, Federico Isaia	Ingeniería en computación	40.574.892

**2024**

## Índice

Programación de módulos para el kernel de linux.....	3
Módulos kernel.....	3
Primeras tareas.....	3
Instalación de Headers.....	4
Mi primer módulo.....	4
Generación, añadido y eliminación del Kernel.....	5
El sistema de ficheros /proc.....	7
¿Cuales módulos posee mi kernel?.....	8
Firma de kernels.....	9
Desafíos del trabajo práctico.....	10
Desafío #1 - Mejoras de seguridad.....	10
Desafío #2 - Check install.....	11
Desafío #3 - Módulos vs programas.....	12
¿Cómo empiezan y terminan unos y otros ?.....	12
¿Qué funciones tiene disponible un programa y un módulo ?.....	12
Espacio de usuario vs espacio del kernel.....	13
Espacio de datos y espacio de código para usuario y kernel.....	13
Analizar posibles riesgos. Drivers. Investigar contenido de /dev.....	13
Desafío #4 - Preguntas adicionales.....	14
Bibliografía.....	15

# Programación de módulos para el kernel de linux

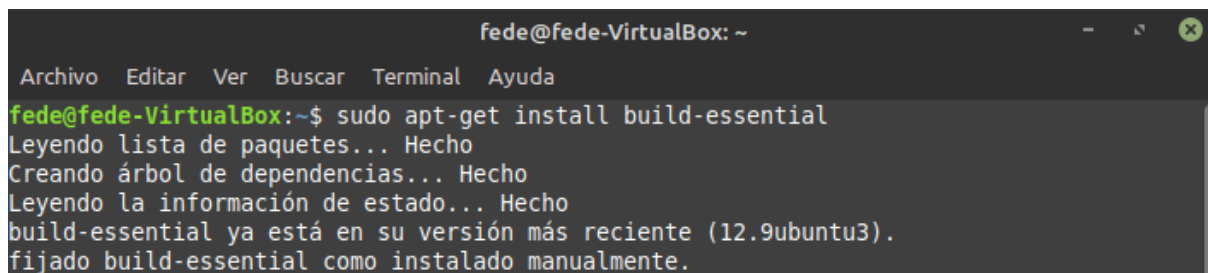
## Módulos kernel

En los kernels de Linux, **los módulos son piezas de código que se pueden cargar dinámicamente en el núcleo del sistema operativo**. Estos módulos amplían la funcionalidad del kernel sin necesidad de recompilarlo o reiniciar el sistema.

**Estos permiten añadir nuevas características, controladores de dispositivos o sistemas de archivos al kernel en tiempo de ejecución sin modificar el código fuente del kernel base.** Se cargan en la memoria solo cuando son necesarios, lo que permite un uso más eficiente de los recursos del sistema. Los usuarios pueden cargar y descargar módulos según sea necesario, lo que permite una configuración más flexible del sistema. Al separar el código en módulos, se facilita el mantenimiento y la actualización del kernel, ya que no es necesario recompilar todo el kernel para hacer cambios en un módulo específico. Además, los módulos facilitan la depuración y el desarrollo de nuevas características del kernel, ya que los cambios pueden probarse de forma incremental sin afectar al kernel en funcionamiento.

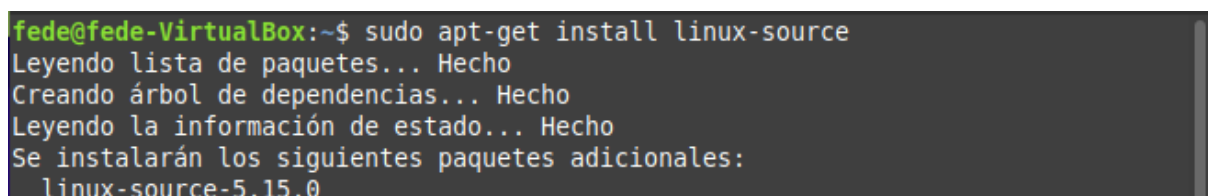
## Primeras tareas

El primer comando permite instalar un conjunto básico de herramientas de compilación llamado **build-essential**, que incluye compiladores y otros programas necesarios para compilar software desde el código fuente.

A terminal window titled 'fedede@fedede-VirtualBox: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The terminal shows the command 'sudo apt-get install build-essential' and its output: 'Leyendo lista de paquetes... Hecho', 'Creando árbol de dependencias... Hecho', 'Leyendo la información de estado... Hecho', 'build-essential ya está en su versión más reciente (12.9ubuntu3).', and 'fijado build-essential como instalado manualmente.'

```
fedede@fedede-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
fedede@fedede-VirtualBox:~$ sudo apt-get install build-essential  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
build-essential ya está en su versión más reciente (12.9ubuntu3).  
fijado build-essential como instalado manualmente.
```

El segundo comando permite **instalar el código fuente del kernel de Linux para permitirnos compilar el kernel desde cero**, modificar la configuración del kernel o compilar módulos del kernel que no están incluidos en la distribución estándar del mismo.

A terminal window titled 'fedede@fedede-VirtualBox: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The terminal shows the command 'sudo apt-get install linux-source' and its output: 'Leyendo lista de paquetes... Hecho', 'Creando árbol de dependencias... Hecho', 'Leyendo la información de estado... Hecho', and 'Se instalarán los siguientes paquetes adicionales: linux-source-5.15.0'.

```
fedede@fedede-VirtualBox:~$ sudo apt-get install linux-source  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
linux-source-5.15.0
```

## Instalación de Headers

Para construir software que interactúe con el kernel, como módulos del kernel o programas de usuario que hagan uso de las llamadas al sistema del kernel, es necesario tener instalados los "headers" correspondientes al kernel en uso. Estos "headers" proporcionan las definiciones necesarias para que el compilador pueda operar correctamente el software y establecer la comunicación adecuada con el kernel.

```
fede@fede-VirtualBox:~$ apt-cache search linux-headers- `uname -r`
linux-headers-5.15.0-106-generic - Linux kernel headers for version 5.15.0 on 64
bit x86 SMP
```

```
fede@fede-VirtualBox:~$ sudo apt-get install kmod linux-headers-5.15.0-106-generic
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
kmod ya está en su versión más reciente (29-lubuntu1).
linux-headers-5.15.0-106-generic ya está en su versión más reciente (5.15.0-106.116).
fijado linux-headers-5.15.0-106-generic como instalado manualmente.
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 14 no actualizados.
```

## Mi primer módulo

Generamos el siguiente archivo .c que servirá como primer módulo

```
/*
2 * hello-1.c - The simplest kernel module.
3 */
4#include <linux/module.h> /* Needed by all modules */
5#include <linux/printk.h> /* Needed for pr_info() */
6
7int init_module(void)
8{
9    pr_info("Hello world 1.\n");
10
11    /* A non 0 return means init_module failed; module can't be
12    loaded. */
13    return 0;
14
15void cleanup_module(void)
16{
17    pr_info("Goodbye world 1.\n");
18}
19
20MODULE_LICENSE("GPL");
```

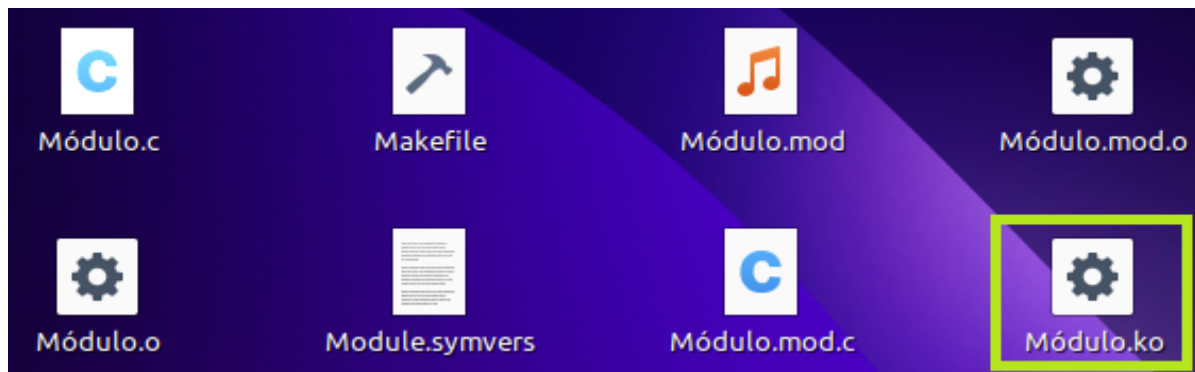
También necesitaremos el makefile correspondiente para compilar nuestro módulo.

```
obj-m += hello-1.o
2
3PWD := $(CURDIR)
4
5all:
6    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
7
8clean:
9    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

### Generación, añadido y eliminación del Kernel

Generamos los archivos mediante el Makefile anteriormente mostrado mediante el siguiente comando:

```
fed@fed-VirtualBox:~/Escritorio$ make
make -C /lib/modules/5.15.0-106-generic/build M=/home/fede/Escritorio/modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-106-generic'
CC [M] /home/fede/Escritorio/Módulo.o
MODPOST /home/fede/Escritorio/Module.symvers
CC [M] /home/fede/Escritorio/Módulo.mod.o
LD [M] /home/fede/Escritorio/Módulo.ko
BTF [M] /home/fede/Escritorio/Módulo.ko
```



Mediante el comando “**modinfo Módulo.ko**” conseguimos información de este archivo. Se compara

```
fed@fed-VirtualBox:~/Escritorio$ modinfo Módulo.ko
filename:      /home/fede/Escritorio/Módulo.ko
license:      GPL
srcversion:    93D1874613E2633FD08C40F
depends:
retpoline:    Y
name:         Módulo
vermagic:     5.15.0-106-generic SMP mod unload modversions
```

```

abel@abel-asus:~$ cd Desktop/
abel@abel-asus:~/Desktop$ modinfo hello-1.ko
filename:       /home/abel/Desktop/hello-1.ko
license:        GPL
srcversion:     93D1874613E2633FD08C40F
depends:
retpoline:      Y
name:           hello_1
vermagic:       6.5.0-35-generic SMP preempt mod_unload modversions
abel@abel-asus:~/Desktop$

```

Los archivos generados con el archivo “Makefile” los cuales tienen extensión .ko (kernel object) son módulos del kernel. Se observa que la única diferencia entre los mismos es el campo denominado “vermagic”. Este campo muestra el nombre de la versión exacta del kernel con la que el módulo fué compilado. Para el primer caso tenemos la versión 5.15.0-106-generic SMP mod\_unload modversions, mientras que para el segundo se tiene la versión 6.5.0-35-generic SMP preempt mod\_unload modversions. Ambas versiones de kernel soportan SMP (Symmetric Multiprocessing). Para el caso 2 tenemos que esta versión es preemptible, lo que significa que el kernel puede suspender la ejecución de una tarea en modo kernel (mientras se está ejecutando código del mismo) para permitir que otra tarea con mayor prioridad se ejecute. Se mejora la capacidad de respuesta y se reduce la latencia. Estas características analizadas (soporte SMP y preempt) son dos características pueden afectar la compatibilidad a la hora de ejecutar un módulo.

Con el comando “**insmod**” lo agregamos al kernel. Si intentamos agregarlo de nuevo, la terminal nos indica que ya existe el archivo y mediante “**rmmod**” puede eliminarse del kernel.

```

fed@fed-VirtualBox:~/Escritorio$ sudo insmod Módulo.ko
fed@fed-VirtualBox:~/Escritorio$ sudo insmod Módulo.ko
insmod: ERROR: could not insert module Módulo.ko: File exists
fed@fed-VirtualBox:~/Escritorio$ sudo rmmod Módulo.ko

```

Por último podemos verificar la correcta instalación mediante “**lsmod | grep Módulo**”.

```

fed@fed-VirtualBox:~/Escritorio$ lsmod | grep Módulo
Módulo          16384  0

```

## El sistema de ficheros /proc

En Linux, hay un mecanismo adicional para que el kernel y los módulos del kernel envíen información a los procesos: el sistema de archivos /proc el cual fue originalmente diseñado para permitir un fácil acceso a la información sobre los procesos (De allí su nombre).

Su funcionamiento es muy similar al utilizado con los controladores de dispositivos: **se crea una estructura con toda la información necesaria para el archivo /proc**, incluidos los punteros a cualquier función de controlador (en nuestro caso, solo uno, el que se llama cuando alguien intenta leer desde el archivo /proc). Luego, **init\_module registra la estructura con el kernel y cleanup\_module la elimina**.

A diferencia de un sistema de archivos convencional que almacena archivos en el disco, /proc almacena información en la memoria RAM y se genera dinámicamente por el kernel. Algunas de los archivos de /proc más utilizados son:

- **cpuinfo y meminfo:** El primero proporciona información detallada sobre el procesador tales como fabricante, modelo, velocidad y características. El segundo muestra información de la memoria del sistema, incluyendo la memoria física total, libre y utilizada.

```
fede@fede-VirtualBox:~$ cd /proc
fede@fede-VirtualBox:/proc$ cat cpuinfo
processor       : 0
vendor_id      : AuthenticAMD
cpu family     : 25
model          : 80
model name     : AMD Ryzen 5 5600G with Radeon Graphics
stepping       : 0
cpu MHz        : 3892.686
cache size     : 512 KB
```

```
fede@fede-VirtualBox:/proc$ cat meminfo
MemTotal:      10995796 kB
MemFree:       7960056 kB
MemAvailable:  9933520 kB
Buffers:       80168 kB
Cached:        2080016 kB
```

- **loadavg:** Muestra el promedio de carga del sistema en diferentes intervalos de tiempo, lo que nos permite monitorear la carga del sistema y la utilización de recursos.

```
fede@fede-VirtualBox:~$ cat /proc/loadavg
0.06 0.05 0.01 1/427 3950
```

- **partitions:** Proporciona una lista las particiones de disco en el sistema, incluyendo información como el tamaño de las particiones y el tipo de sistema de archivos.

```
fede@fede-VirtualBox:~$ cat /proc/partitions
major minor #blocks name

11        0      52244 sr0
 8         0  52428800 sda
 8         1     1024 sda1
 8         2   525312 sda2
 8         3  30928896 sda3
```

- **version:** Muestra información sobre la versión del kernel de Linux que está siendo ejecutada en el sistema, así como también información adicional sobre la configuración del kernel.

```
fede@fede-VirtualBox:~$ cat /proc/version
Linux version 5.15.0-106-generic (build@lcy02-amd64-017) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #116-Ubuntu SMP Wed Apr 17 09:17:56 UTC 2024
```

- **net:** Proporciona información sobre el estado de las conexiones de red del sistema. En particular, el archivo `/proc/net/tcp` nos muestra información detallada sobre las conexiones TCP activas, como las direcciones IP y puertos locales y remotos, el estado de la conexión y el número de identificación del proceso (PID) asociado.

```
fede@fede-VirtualBox:~$ cat /proc/net/tcp
sl  local_address rem_address  st tx_queue rx_queue tr tm->when retrnsmt
0: 3500007F:0035 00000000:0000 0A 00000000:00000000 00:00000000 00000000
1: 0100007F:0277 00000000:0000 0A 00000000:00000000 00:00000000 00000000
```

Se ejecuta el módulo `hwninfo` en otra computadora (se genera la misma información detallada anteriormente) y se genera un archivo de texto con la información obtenida.

```
abel@abel-asus:~/Desktop/Abel/Facu/Sistemas de Computación/Repositorio SC/SistComp/TP4/Modulos/hw_inf
o$ hwninfo > hw_info.txt
abel@abel-asus:~/Desktop/Abel/Facu/Sistemas de Computación/Repositorio SC/SistComp/TP4/Modulos/hw_inf
o$
```

Se adjunta el URL del archivo “hw\_info.txt” URL: ().

### ¿Cuales módulos posee mi kernel?

Para encontrar qué módulos se encuentran en mi kernel actualmente podemos utilizar el comando “`lsmod`”, con el cual encontramos que hay más de 80 módulos disponibles en el kernel actualmente.

```
fede@fede-VirtualBox:~$ sudo lsmod
[sudo] contraseña para fede:
Module              Size  Used by
ufs                  106496  0
qnx4                  16384  0
hfsplus              118784  0
hfs                   65536  0
minix                 49152  0
ntfs                  122880  0
msdos                 20480  0
jfs                   233472  0
xfs                   1769472  0
isofs                 53248  1
binfmt_misc          24576  1
zfs                   4112384  6
```



Se generan archivos de texto para la lista generada para las computadoras de los integrantes con el comando `lsmod` (módulos de kernel actuales). Se ejecuta la siguiente línea de comando en la terminal.

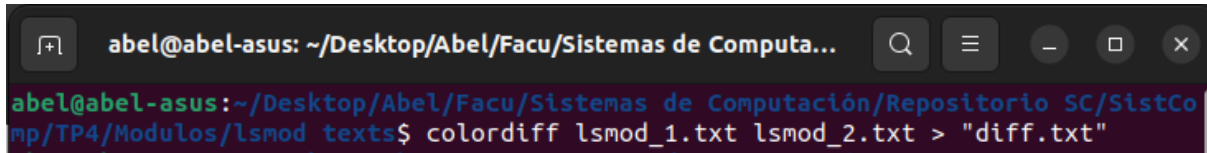
```
abel@abel-asus:~$ lsmod > /home/abel/Desktop/lsmod_2.txt
abel@abel-asus:~$
```

Se realiza la diferencia entre los archivos de texto que contienen los módulos de kernel para los dos casos de estudio.

```
abel@abel-asus: ~/Desktop/Abel/Facu/Sistemas de Computación/Repositorio SC/SistCo...
abel@abel-asus:~/Desktop/Abel/Facu/Sistemas de Computación/Repositorio SC/SistComp/TP4/Modulos/lsmod
mp/TP4/Modulos/lsmod texts$ colordiff lsmod_1.txt lsmod_2.txt
2,11c2,47
< ufs                106496  0
< qnx4                16384  0
< hfsplus            118784  4
< hfs                 65536  0
< manix              49152  0
< nfts               122880  0
< msdos              20480  0
< jfs                233472  0
< xfs                1769472  0
< isoofs             53248  1
--
> mimodulo           12288  0
> snd_seq_dummy       12288  0
> rfcomm              98304  4
> usbhid              77824  0
> vboxnetadp          28672  0
> vboxnetflt          36864  0
> vboxdrv             692224  2 vboxnetadp,vboxnetflt
> ccm                 20480  3
> cmac                12288  2
> algif_hash          12288  1
> algif_skcipher      12288  1
> af_alg              32768  6 algif_hash,algif_skcipher
> bnep                32768  2
> snd_sof_pci_intel_tgl 12288  0
> snd_hda_codec_hdmi  94208  1
> snd_sof_intel_hda_common 200704 1 snd_sof_pci_intel_tgl
> soundwire_intel     65536  1 snd_sof_intel_hda_common
> snd_sof_intel_hda_mlink 45056 2 soundwire_intel,snd_sof_intel_hda_common
> soundwire_cadence   40960  1 soundwire_intel
> snd_sof_intel_hda    24576  1 snd_sof_intel_hda_common
> snd_sof_pci          24576  2 snd_sof_intel_hda_common,snd_sof_pci_intel_tgl
> snd_sof_xtensa_dsp   12288  1 snd_sof_intel_hda_common
> snd_sof             360448  3 snd_sof_pci,snd_sof_intel_hda_common,snd_sof_intel_hda
> snd_sof_utils        16384  1 snd_sof
> snd_soc_hdac_hda     24576  1 snd_sof_intel_hda_common
> snd_hda_ext_core     36864  4 snd_sof_intel_hda_common,snd_soc_hdac_hda,snd_sof_intel_hda_mlink,snd_sof_intel_hda
> snd_soc_acpi_intel_match 94208 2 snd_sof_intel_hda_common,snd_sof_pci_intel_tgl
> snd_soc_acpi         12288  2 snd_soc_acpi_intel_match,snd_sof_intel_hda_common
> soundwire_generic_allocation 12288 1 soundwire_intel
> soundwire_bus        110592 3 soundwire_intel,soundwire_generic_allocation,soundwire_cadence
> snd_hda_codec_realtek 192512 1
> snd_hda_codec_generic 122880 1 snd_hda_codec_realtek
> snd_soc_core         446464 4 soundwire_intel,snd_sof,snd_sof_intel_hda_common,snd_soc_hdac_hda
> snd_compress         28672  1 snd_soc_core
> ac97_bus             12288  1 snd_soc_core
> snd_pcm_dmaengine     16384  1 snd_soc_core
> snd_hda_intel        61440  5
> intel_tcc_cooling     12288  0
> snd_intel_dspcfg      32768  3 snd_hda_intel,snd_sof,snd_sof_intel_hda_common
```

La primera línea de retorno “2,11c2,47”, nos indica que en las líneas de la 2 a la 11 del primer archivo deben ser reemplazadas (indicador de cambio “c”) por líneas 2 a 47 del segundo archivo.

Se genera un archivo de texto de lo obtenido anteriormente.

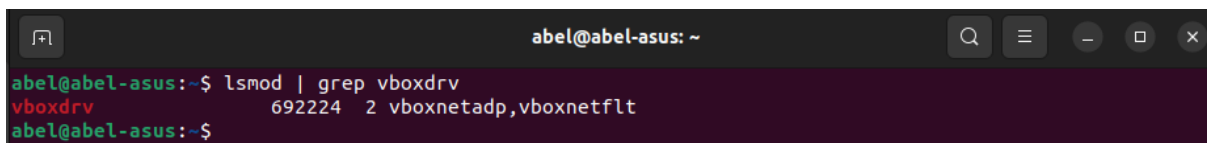


```
abel@abel-asus: ~/Desktop/Abel/Facu/Sistemas de Computa...
abel@abel-asus:~/Desktop/Abel/Facu/Sistemas de Computación/Repositorio SC/SistCo
mp/TP4/Modulos/lsmod texts$ colordiff lsmod_1.txt lsmod_2.txt > "diff.txt"
```

Explicar las diferencias entre los archivos de texto.

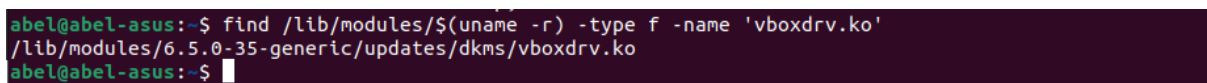
Cuando un módulo de kernel se encuentra cargado pero no disponible quiere decir que el mismo se encuentra en memoria pero su archivo de extensión .ko (kernel object) en disco no está presente. Esto puede ocurrir debido a que se cargó en un momento y luego se eliminó físicamente del sistema de archivos, es decir, no disponible para recarga o reinserción. Otro caso es que un módulo no esté disponible porque se actualizó el kernel o se cambió de directorio. Esto puede ocurrir en el caso que el módulo dependa de otros que ya no estén presentes.

Se puede realizar una verificación del estado de un módulo con dos comandos, uno para verificar la carga y otro para la disponibilidad del mismo. Para verificar la carga se aplica el comando `lsmod | grep` que se ejecutó anteriormente para el módulo de ejemplo. Se realiza implementa para el módulo “vboxdrv”.



```
abel@abel-asus: ~
abel@abel-asus:~$ lsmod | grep vboxdrv
vboxdrv                692224  2 vboxnetadp,vboxnetflt
abel@abel-asus:~$
```

Se verifica que se encuentre en memoria el archivo kernel object (.ko).



```
abel@abel-asus:~$ find /lib/modules/$(uname -r) -type f -name 'vboxdrv.ko'
/lib/modules/6.5.0-35-generic/updates/dkms/vboxdrv.ko
abel@abel-asus:~$
```

En caso que no exista el archivo no se debería tener retorno en el terminal y como solución se propone reinstalar el módulo con el siguiente comando

`sudo apt-get install --reinstall virtualbox-dkms`

En caso que no se encuentre cargado el módulo, puede hacerse en forma manual (sólo si está disponible) con la siguiente línea de comando.

```
sudo modprobe vboxdrv
```

Se muestra ahora un caso en el que no se tiene la carga de los módulos ufs, qnx4, hfsplus (obtenidos del archivo diff).

```
abel@abel-asus:~$ lsmod | grep ufs
abel@abel-asus:~$ lsmod | grep qnx4
abel@abel-asus:~$ lsmod | grep hfsplus
abel@abel-asus:~$
```

En los tres casos no se obtuvo retorno, por lo que no se encuentran cargados.

## Firma de kernels

La firma de módulos en Linux es un mecanismo de seguridad que permite verificar la autenticidad y la integridad de los módulos del kernel antes de cargarlos en el sistema. **Consiste en firmar digitalmente los módulos con una clave privada, y luego verificar esa firma utilizando una clave pública antes de cargar el módulo en el kernel.**

Esto tiene como ventajas:

- Seguridad mejorada: La firma de módulos ayuda a prevenir la carga de módulos maliciosos o modificados que puedan comprometer la seguridad del sistema.
- Integridad del módulo: Permite verificar que un módulo no ha sido alterado desde que fue firmado, garantizando su integridad.
- Control de versiones: Permite asegurar que solo se carguen módulos específicos que han sido firmados para una determinada versión del kernel, evitando problemas de compatibilidad.

A continuación mostramos una firma del archivo **des\_generic.ko** conseguida buscando su información mediante el comando modinfo.

```
signature: 63:F0:73:98:0F:48:A8:7C:76:27:A7:63:32:27:E8:D3:95:9C:02:23:
E8:7D:71:55:BD:75:30:D1:28:FA:95:7D:99:46:1A:69:FF:21:73:68:
F3:51:4C:B6:20:03:A2:6B:0F:2D:D2:FD:AF:16:A9:6A:8D:36:EB:DB:
E0:58:07:EC:CB:B0:D5:A8:79:2F:E0:83:14:E6:09:21:95:89:F2:AF:
50:50:4B:E1:24:9E:CF:04:00:86:A3:55:AF:D0:13:04:0D:17:0F:1F:
05:08:12:34:14:31:5D:B0:7C:57:57:54:C6:E0:91:C2:FF:61:B1:88:
F6:DB:73:21:3F:5A:F6:41:A6:7C:DE:90:4D:E4:7C:E0:C0:E8:0F:21:
96:F3:9A:BA:1B:4E:BD:F8:FE:B2:A9:65:21:9E:B1:4F:00:EB:E4:BA:
EB:FD:09:F8:1D:EC:CB:D4:00:0C:A6:1A:1B:4D:18:33:D1:CF:76:5A:
B9:1B:F4:A3:5D:0C:F5:6F:42:8A:40:4A:28:FB:20:5A:56:05:04:39:
DA:DF:B7:BB:F7:20:45:EE:21:7D:44:1D:7E:07:55:CC:11:59:B0:86:
20:61:F9:ED:D3:1D:59:57:D4:3D:BA:49:2B:EB:D4:95:97:7B:E0:6E:
9E:56:F3:A1:34:4E:8B:34:86:55:57:27:A7:90:49:A6:F1:06:30:8B:
2C:C2:76:FE:3F:37:29:71:18:EB:94:6B:9B:88:DF:63:A0:DB:CA:52:
DD:4F:97:D1:EA:F4:47:35:BF:F8:9C:AE:3F:B9:D8:7B:EF:2B:91:8E:
66:5D:24:A6:36:E0:91:88:07:C9:61:B0:F6:03:B5:08:E0:C6:EA:F9:
50:C9:9A:12:42:78:48:53:CF:45:70:BE:9B:33:59:44:30:04:9A:B9:
A1:E7:54:10:DC:AC:73:1F:E6:D6:D6:51:45:E0:2D:9A:BA:80:D5:6B:
49:16:46:15:B7:A1:F0:FD:CD:BE:EE:9D:E8:41:00:7F:34:01:2D:B4:
30:B7:46:06:B2:B8:1A:59:C1:07:E3:67:41:30:2E:A6:FD:A3:F2:1A:
34:05:3A:6D:3A:4C:6E:42:F5:72:EB:90:DB:A8:07:ED:8E:51:B2:ED:
2A:A9:6F:3F:7D:B6:86:87:AD:54:E9:FA:7A:2D:47:73:F9:98:62:59:
6D:80:E7:F1:22:09:5F:9C:77:D1:A4:8E:AA:CD:32:9D:54:27:CC:71:
BD:CE:E4:9C:8D:37:BD:D6:9E:1A:03:4F:8F:AD:A8:2E:DE:DF:DD:26:
7B:5D:53:ED:83:1A:BD:BC:E5:E6:FA:66:A4:6D:AA:3B:53:50:01:29:
D2:AC:F1:AB:F3:BE:8A:D6:80:A2:BE:8A
```

## **Desafíos del trabajo práctico**

### **Desafío #1 - Mejoras de seguridad**

**Mediante la firma de los módulos, impulsar acciones que permitan mejorar la seguridad del kernel, concretamente: evitando cargar módulos que no estén firmados.**

**¿Qué otras medidas de seguridad podemos implementar para asegurar nuestro ordenador desde el kernel ?**

**La firma de módulos es una técnica que consiste en firmar digitalmente los módulos antes de que puedan ser cargados en el kernel.** Esto asegura que sólo los módulos autenticados y aprobados por una entidad de confianza puedan ser integrados al kernel, evitando así la carga de módulos maliciosos o no autorizados. La idea es generar un par de claves (privada y pública) utilizando una herramienta de criptografía como OpenSSL.

La clave privada se utilizará para firmar los módulos, y la clave pública se integrará en el kernel para verificar las firmas.

Utilizar la herramienta scripts/sign-file del kernel para firmar los módulos con la clave privada.

Otras Medidas de Seguridad para Asegurar el Ordenador desde el Kernel

#### **1. Secure Boot**

Implementar Secure Boot para garantizar que el sistema se inicie solo con software verificado. Esto previene la ejecución de bootloaders, kernels y módulos no autorizados desde el inicio del sistema.

#### **2. SELinux (Security-Enhanced Linux)**

Habilitar y configurar SELinux para aplicar políticas de control de acceso más estrictas. SELinux proporciona un mecanismo de seguridad a nivel de kernel que permite definir políticas para controlar qué recursos puede acceder a cada proceso.

#### **3. Control de Acceso Basado en Roles (RBAC)**

Utilizar RBAC para asignar permisos basados en roles específicos, limitando así las capacidades de los usuarios y procesos según sus funciones y necesidades mínimas.

#### **4. Integridad del Kernel (Kernel Integrity)**

Utilizar tecnologías como IMA (Integrity Measurement Architecture) y EVM (Extended Verification Module) para garantizar la integridad del sistema de archivos y los binarios del kernel.

Creamos el archivo x509.genkeys con el contenido que menciona la guía mencionada dentro de la carpeta donde se encuentran los archivos de compilación del kernel, en este caso es en: /lib/modules/5.15.0-106-generic/build/certs con el siguiente contenido.

```
FILE x509.genkey Key generation configuration file

[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
CN = Modules

[ myexts ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
```

### ¿Qué pasa si mi compañero con Secure Boot habilitado intenta cargar un módulo firmado por mí?

Si otra persona con Secure Boot habilitado intenta cargar un módulo firmado por otro, es probable que el proceso falle debido al Secure Boot. Este último es una característica de seguridad diseñada para proteger el proceso de arranque del sistema operativo. La tarea fundamental de esta herramienta de seguridad es la verificación de la firma digital de los componentes cargados durante el arranque, incluidos los módulos del kernel.

Al intentar cargar un módulo en condiciones de una firma particular se tiene que el Secure Boot generará un mensaje de advertencia o error en el registro del sistema operativo o en el proceso de arranque. Otro caso es que directamente se rechace la misma, ya que esta herramienta sólo permite cargar módulos firmados con claves que estén firmadas por una autoridad de confianza.

Si la clave pública correspondiente a la firma del módulo está presente en la lista de claves autorizadas del firmware UEFI, entonces el módulo se cargará correctamente.

En algunos sistemas se puede configurar el Secure Boot en un "Modo de usuario" donde se pueden cargar módulos firmados por el usuario. Sin embargo, esto

generalmente requiere la intervención del usuario para agregar manualmente las claves públicas correspondientes a las firmas de los módulos.

## Desafío #2 - Check install

**¿Qué es checkinstall y para qué sirve? Empaquetar un hello world utilizándolo. Revisar la bibliografía para impulsar acciones que permitan mejorar la seguridad del kernel, concretamente: evitando cargar módulos que no estén firmados.**

Es una herramienta que se utiliza para crear paquetes de instalación a partir de la compilación de software desde su código fuente. Esto facilita la instalación, desinstalación y gestión del software, ya que el paquete creado se puede administrar a través del gestor de paquetes del sistema.

En particular este programa soporta los formatos .DEB (Debian y sus derivados) .RPM (RHEL) y paquetes Slackware.

Procedemos a instalarla como se muestra a continuación:

```
fede@fede-VirtualBox:~/Escritorio$ sudo apt-get install checkinstall
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  checkinstall
```

### Ejemplo simple con un programa "Hello world"

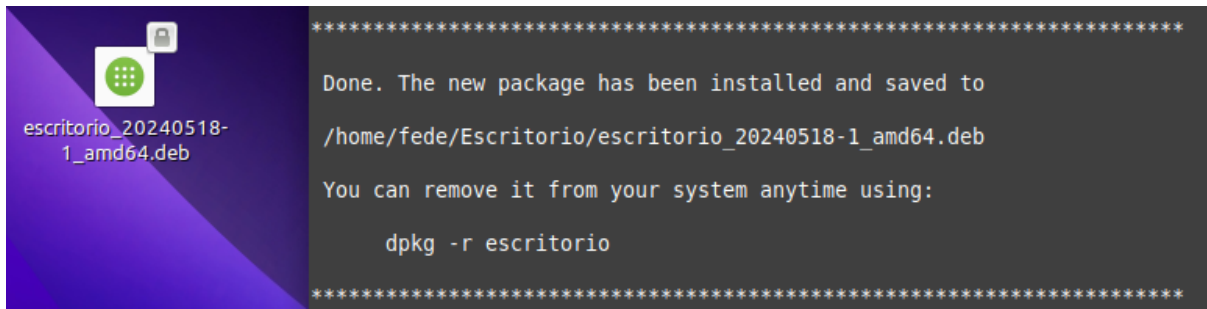
En el folder donde se ejecuta el comando "checkinstall" debe estar presente el código fuente y el archivo Makefile para compilar el project, es importante que haya un target "install".clear

```
fede@fede-VirtualBox:~/Escritorio$ sudo checkinstall

checkinstall 1.6.3, Copyright 2010 Felipe Eduardo Sanchez Diaz Duran
  Este software es distribuido de acuerdo a la GNU GPL

The package documentation directory ./doc-pak does not exist.
Should I create a default set of package docs?  [y]: y

Preparando la documentación del paquete...OK
```



En la imagen se observa que el checkinstall instaló el software, pero también creó un paquete .deb.



## Desafío #3 - Módulos vs programas

### ¿Cómo empiezan y terminan unos y otros ?

**Un programa típico comienza con una función `main()`**, ejecuta una serie de instrucciones y termina después de completar estas instrucciones, sin embargo los módulos del kernel siguen un patrón diferente.

Un módulo siempre **comienza con la función `init_module`** o una función designada por la llamada `module_init`. **Esta función actúa como el punto de entrada del módulo, informando al kernel de las funcionalidades del módulo y preparando al kernel para utilizar las funciones del módulo cuando sea necesario.** Después de realizar estas tareas, la función de entrada regresa y el módulo permanece inactivo hasta que el kernel requiere su código.

**Todos los módulos concluyen invocando `cleanup module`** o una función especificada a través de la llamada `module_exit`. Esto sirve como la función de salida del módulo, revirtiendo las acciones de la función de entrada al anular el registro de las funcionalidades previamente registradas.

**Es obligatorio que cada módulo tenga tanto una función de entrada como una de salida.**

### ¿Qué funciones tiene disponible un programa y un módulo ?

Al desarrollar software, los programadores a menudo utilizan funciones que no han definido ellos mismos. Estas funciones son parte de las bibliotecas estándar del lenguaje de programación que están utilizando. Por ejemplo, el programador aunque no haya definido la función `printf()` en su programa, puede usarla porque el compilador sabe dónde encontrar su código, dentro de la biblioteca estándar de C.

Por otro lado, cuando se desarrolla un módulo para el kernel, no se tiene acceso a las funciones de las bibliotecas estándar de C, como `printf()`, porque estas funciones están diseñadas para ejecutarse en el espacio de usuario, no en el kernel.

Por lo que en su lugar, estos módulos utilizan las funciones proporcionadas por el propio kernel. Por ejemplo, en un simple módulo de "hello world" para el kernel de Linux, se utiliza la función `pr_info()` para imprimir un mensaje en el registro del kernel. Esta función no está definida en una biblioteca estándar de C, sino que está definida en el propio kernel.

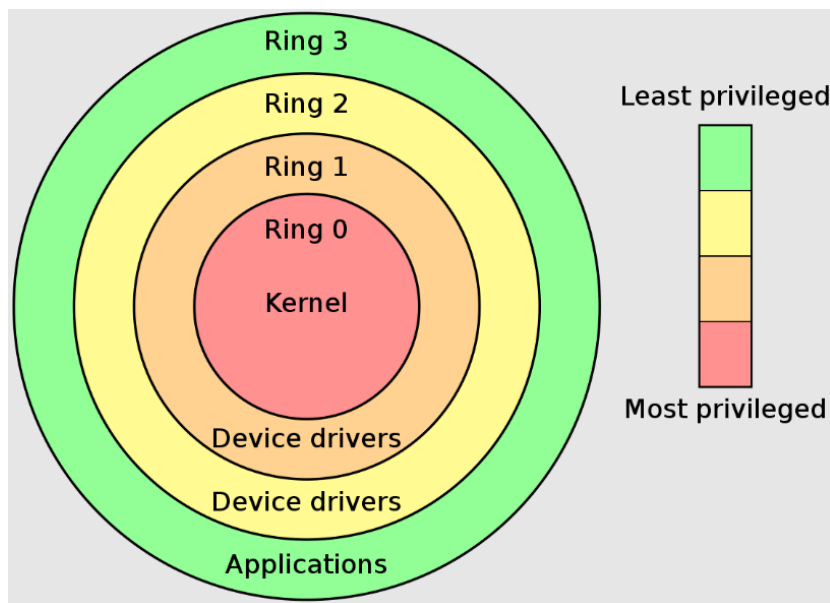
**En resumen, mientras que en el espacio de usuario los programadores pueden utilizar funciones de bibliotecas estándar como `printf()` porque el compilador sabe dónde encontrar su código, en el kernel del sistema operativo solo se pueden utilizar las funciones que proporciona el propio kernel. Esto se debe a que el kernel y el espacio de usuario son entornos de ejecución diferentes, con sus propias reglas y restricciones.**

### Espacio de usuario vs espacio del kernel.

El kernel se encarga principalmente de gestionar el acceso a los recursos, ya sea una tarjeta de video, un disco duro o memoria. Los programas a menudo compiten por los mismos recursos. El papel del kernel es mantener el orden, asegurando que los usuarios no accedan a los recursos indiscriminadamente.

Para gestionar esto, las CPUs operan en diferentes modos, cada uno ofreciendo diferentes niveles de control del sistema. La arquitectura Intel 80386 presentaba cuatro modos diferentes, conocidos como anillos. Sin embargo, Unix utiliza solo dos de estos anillos: **el anillo más alto (anillo 0: "modo supervisor", donde todas las acciones son permitidas) y el anillo más bajo, conocido como "modo usuario".**

Normalmente, usar **una función de biblioteca se realiza en modo usuario. La misma invoca a una o más llamadas al sistema, y estas llamadas al sistema se ejecutan en nombre de la función de biblioteca, pero lo hacen en modo supervisor** ya que forman parte del propio kernel. Una vez que la llamada al sistema completa su tarea, retorna y la ejecución se transfiere de nuevo al modo usuario.



### Espacio de datos y espacio de código para usuario y kernel. Posibles riesgos.

Cuando se crea un proceso, el kernel reserva una porción de memoria física real y se la entrega al proceso para que la use en su código ejecutable y almacene los datos del proceso: Variables, stack y heap, entre otras cosas.

Esta memoria comienza en 0x00000000 y se extiende hasta lo que sea necesario. **Dado que el espacio de memoria de dos procesos cualesquiera no se superpone, cada proceso que puede acceder a una dirección de memoria estaría accediendo a una ubicación diferente en la memoria física real..**

Por otra parte, el kernel también tiene su propio espacio de memoria. Dado que un módulo es código que puede insertarse y eliminarse dinámicamente en el kernel comparte el espacio de código del kernel en lugar de tener el suyo propio. Por lo tanto, **si tu módulo provoca un fallo de segmentación, el kernel también falla. Y si empiezas a sobrescribir datos debido a un error de desbordamiento de índice, estarás pisoteando datos (o código) del kernel.** Esto se aplica a cualquier sistema operativo que utilice un kernel monolítico.

### Drivers. Investigar contenido de /dev.

Un tipo común de módulo en el kernel es el controlador de dispositivos, que proporciona funcionalidad para hardware como un puerto serial.

Cada pieza de hardware está representada por un archivo ubicado en /dev llamado **archivo de dispositivo**, que proporciona los medios para comunicarse con el hardware. El controlador de dispositivos maneja la comunicación en nombre de un programa de usuario.

Estos archivos de dispositivos utilizan como parte de la descripción del dispositivo:

- Número mayor (major device number): Es un identificador para el driver que controla al dispositivo.
- Número menor (minor device number): Este número se agrega para identificar a cada dispositivo controlado por un driver en particular, ya que un mismo driver puede controlar a varias piezas de hardware.

### Ejemplo de Archivos de Dispositivos

```
fede@fede-VirtualBox:~$ ls -l /dev/sd*  
brw-rw---- 1 root disk 8, 0 may 18 11:54 /dev/sda  
brw-rw---- 1 root disk 8, 1 may 18 11:54 /dev/sda1  
brw-rw---- 1 root disk 8, 2 may 18 11:54 /dev/sda2  
brw-rw---- 1 root disk 8, 3 may 18 11:54 /dev/sda3
```

Los números separados por una coma son el número mayor y menor del dispositivo. El número mayor indica qué controlador se usa para acceder al hardware. Todos los archivos de dispositivo con el mismo número mayor son controlados por el mismo controlador. En el ejemplo anterior, todos los números mayores son 8, porque todos están controlados por el mismo controlador.

El número menor es usado por el controlador para distinguir entre los diferentes dispositivos que controla. Aunque los tres dispositivos son manejados por el mismo controlador, tienen números menores únicos porque el controlador los ve como diferentes piezas de hardware.

### Tipos de Dispositivos

**Los dispositivos se dividen en dos tipos: dispositivos de caracteres y dispositivos de bloques.**

- Los dispositivos de bloques tienen un búfer para solicitudes, lo que les permite elegir el mejor orden para responder a las solicitudes. Esto es importante en el caso de dispositivos de almacenamiento, donde es más rápido leer o escribir sectores que están cerca uno del otro.
- Los dispositivos de bloques solo aceptan y devuelven datos en bloques, mientras que los dispositivos de caracteres pueden usar tantos o tan pocos bytes como necesiten

La mayoría de los dispositivos son de caracteres porque no necesitan este tipo de almacenamiento en búfer y no operan con un tamaño de bloque fijo. Puedes saber si un archivo de dispositivo es de bloques o de caracteres mirando el primer carácter en la salida de `ls -l`. **Si es 'b', es un dispositivo de bloques, y si es 'c', es un dispositivo de caracteres.** Los dispositivos anteriores son dispositivos de bloques.

Ejemplo de algunos dispositivos de caracteres (los puertos seriales):

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
fed@fed-VirtualBox:~$ ls -l /dev/ttyS[0-3]
crw-rw---- 1 root dialout 4, 64 may 18 11:54 /dev/ttyS0
crw-rw---- 1 root dialout 4, 65 may 18 11:54 /dev/ttyS1
crw-rw---- 1 root dialout 4, 66 may 18 11:54 /dev/ttyS2
crw-rw---- 1 root dialout 4, 67 may 18 11:54 /dev/ttyS3
```

## Bibliografía

- [https://access.redhat.com/documentation/es-es/red\\_hat\\_enterprise\\_linux/8/html/managing\\_monitoring\\_and Updating\\_the\\_kernel/signing-kernel-modules-for-secure-boot\\_managing-kernel-modules](https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/8/html/managing_monitoring_and Updating_the_kernel/signing-kernel-modules-for-secure-boot_managing-kernel-modules)
- <https://sysprog21.github.io/lkmpg/#what-is-a-kernel-module>
- <https://opensource.com/article/19/10/strace>
- [https://docs.google.com/presentation/d/1BYES6Zkfx5K85REWyXsFeW-VngBLOzIDzaYCsTVoc0Y/edit#slide=id.g724a4c87a0\\_0\\_5](https://docs.google.com/presentation/d/1BYES6Zkfx5K85REWyXsFeW-VngBLOzIDzaYCsTVoc0Y/edit#slide=id.g724a4c87a0_0_5)