



Curso: Sistemas Operativos

Profesora: Mireya Paredes

Reporte 2: Manejo de módulos en Linux

Miembros del equipo:

Rafael Agustín López Hernández – 160805

Alexander Díaz Ruiz - 160046

Jose Ashamat Jaimes Saavedra- 158320

Introducción

Durante la clase anterior, vimos que hay diferentes tipos de estructuras de sistemas operativos, como modular, monolítico, microkernel, etc. Linux es considerado un sistema operativo modular por su sistema de carga de programas a través de módulos. El objetivo de esta práctica es familiarizarse con el sistema de manejo de módulos de linux.

En esta práctica se hará un kernel module y se deberá cargarlo en el kernel de Linux. Se usará un editor de código C y la terminal para correr el código y poder poner comandos de consola.

En el libro de Sistemas Operativos de Silberschatz podemos encontrar más información y resolver algunos problemas con la programación del siguiente.

Desarrollo

Primero, se crearon los programas con el comando de linux *touch* y se sobrescribe con el editor de código *nano*, cual está integrado en el shell.

Al inicio se intentó hacer en el distro (distribución) de linux *manjaro*, pero los headers de la carpeta linux no existían y se tuvo que hacer en el distro *Ubuntu*.

Parte I

La primera parte de la práctica consistió en en crear e insertar un módulo al kernel de Linux.

Un módulo es esencialmente una extensión del kernel, el cual brinda servicios adicionales a los ya predefinidos de manera dinámica mientras el kernel está corriendo (Silberschatz, 2015).

Para ver todos los módulos cargados al kernel, se utiliza el comando `lsmod`, el cual enlista el nombre de cada módulo, su tamaño, y el número de veces que está siendo utilizado por programas en ejecución, así como los nombres de éstos.

La *Figura 1* a continuación muestra un ejemplo:

```
alex@alex-VirtualBox:~$ lsmod
Module              Size  Used by
edac_mce_amd        28672  0
crct10dif_pclmul    16384  1
crc32_pclmul        16384  0
ghash_clmulni_intel 16384  0
aesni_intel         372736 0
snd_intel8x0        45056  2
aes_x86_64          20480  1 aesni_intel
snd_ac97_codec      135168 1 snd_intel8x0
crypto_simd         16384  1 aesni_intel
cryptd              24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
ac97_bus            16384  1 snd_ac97_codec
snd_pcm             102400 2 snd_intel8x0,snd_ac97_codec
glue_helper         16384  1 aesni_intel
vmwgfx              290816 2
snd_seq_midi        20480  0
joydev              28672  0
snd_seq_midi_event  16384  1 snd_seq_midi
ttm                 102400 1 vmwgfx
snd_rawmidi         36864  1 snd_seq_midi
drm_kms_helper      180224 1 vmwgfx
snd_seq             69632  2 snd_seq_midi,snd_seq_midi_event
drm                 479232 5 vmwgfx,drm_kms_helper,ttm
fb_sys_fops         16384  1 drm_kms_helper
snd_seq_device      16384  3 snd_seq,snd_seq_midi,snd_rawmidi
syscopyarea         16384  1 drm_kms_helper
snd_timer           36864  2 snd_seq,snd_pcm
input_leds          16384  0
```

Figura 1. Proyección del comando `lsmod` en Bash.

Para el propósito de este ejercicio, se empleó un módulo de kernel sumamente sencillo, el cual imprime un mensaje una vez que éste haya sido cargado o removido del kernel.

Su código en el lenguaje C se guardó en un programa titulado `simple.c`, el cual se muestra a continuación:

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

/* Module entry point
   This function is called when the module is loaded. */
int simple_init(void)
{
    // printk() is the kernel equivalent of printf()
    /* KERN_INFO is the function's priority flag: an informational message
       */
    printk(KERN_INFO "Loading Module\n");

    /* If the function returns anything other than 0, it represents failure.
       */
    return 0;
}

/* Module exit point
   This function is called when the module is removed. */
void simple_exit(void)
```

```
/* Module exit point
   This function is called when the module is removed. */
void simple_exit(void)
{
    printk(KERN_INFO "Removing Module\n");
}

/* Macros for registering module entry and exit points. */
module_init(simple_init);
module_exit(simple_exit);

/* Represent details regarding the software license, description of the module, and author.
   */
MODULE_LICENSE("GPL");
```

```
/* Represent details regarding the software license, description of the module, and author.
   */
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("SGG");
```

Figuras 2, 3, 4. Contenido del programa simple.c, visto desde el editor de texto Nano.

El archivo de cabecera `<linux/init>.h` permite la inicialización de módulos, implementando los macros `module_init()` y `module_exit()`, los cuales ejecutan la función que les es otorgada como parámetro al momento de cargar y de remover un módulo del kernel, respectivamente. Cabe mencionar que es necesario definir las funciones empleadas como argumento de los macros antes de poder llamar a estos últimos.

`<linux/kernel>.h` importa consigo la expansión macro de la función `printk()`, la cual sirve el mismo propósito que `printf()`, con la única excepción de que imprime su salida a un buffer del log del kernel en lugar de a la terminal. Para poder leer sus contenidos, se tendrá que invocar el comando `dmesg`.

Finalmente, la cabecera `<linux/module>.h` necesita ser incluido en cada módulo. Provee el programa con los macros `MODULE_DESCRIPTION()` y `MODULE_AUTHOR()`, los cuales, aunque no empleados por el mismo kernel, tienen un uso estándar para su documentación.

La primera función que se define en el programa es `simple_init()`, el cual no recibe parámetros y regresa un entero tras ser llamado.

La función invoca a `printk()`, indicando a través de su bandera de prioridad `KERN_INFO` de que se trata de un mensaje informativo, el cual imprime la cadena de caracteres "Loading Module".

Si la función corre sin problemas, regresará un 0.

La segunda y última función a emplear es `simple_exit()`, que no acepta parámetros ni produce una salida.

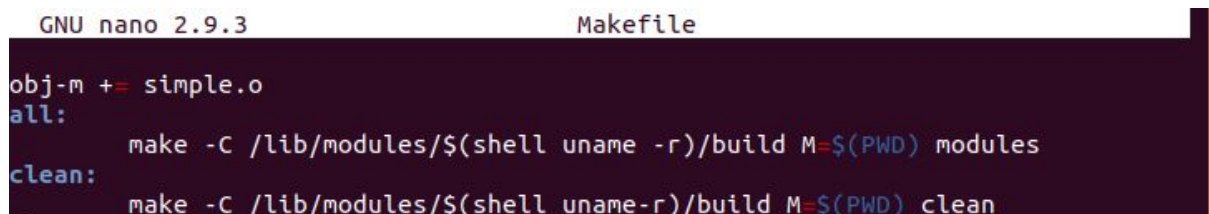
Similar a `simple_init()`, esta función únicamente imprimirá un mensaje informativo: "Removing Module".

Seguido de las funciones se llama a los macros `module_init()` y `module_exit()`, indicando que `simple_init()` será la función a ejecutar al momento en que se cargue el módulo al kernel, y `simple_exit()` cuando se remueva.

Para compilar el módulo, se empleará lo que se conoce como un `Makefile`.

Un `Makefile` es un programa que simplifica ejecutables de programas que requieren varios módulos. Así, en lugar de compilar todos los módulos usados por un programa complejo cada vez que se realice un cambio, sólo se necesitará compilar el `Makefile` una vez antes de poder ser ejecutado nuevamente.

El `Makefile` a utilizar en la práctica viene preestablecido, y su contenido se demuestra en la siguiente *Figura 5*:



```
GNU nano 2.9.3 Makefile
obj-m += simple.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Figura 5. Contenido del `Makefile` responsable de compilar el módulo `simple.c`

Para compilar el módulo a través de la terminal, se ocupa el comando `make`. Una vez ejecutado, se producirán varios archivos en el directorio de trabajo actual, incluyendo `simple.ko`: el módulo de kernel compilado:


```
alex@alex-VirtualBox:~$ make
make -C /lib/modules/5.0.0-23-generic/build M=/home/alex modules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-23-generic'
  CC [M] /home/alex/simple.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC /home/alex/simple.mod.o
  LD [M] /home/alex/simple.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-23-generic'
```

```
alex@alex-VirtualBox:~$ ls
Desktop      Graph.py.save  Music      separate_process.c  simple.mod.o
Documents    Makefile       Pictures   simple.c             simple.o
Downloads    modules.order  programa   simple.ko            Templates
eg_clase.c   Module.symvers Public       simple.mod.c         Videos
```

Figuras 6, 7. Ejecución del comando `make` en la terminal, así como un enlistado del directorio actual en donde se aprecia la adición consecuente de `simple.ko`

Una vez compilado el módulo, éste se puede subir al kernel a través del comando `insmod`, el cual requiere permisos de administrador.

Hecho esto, al ejecutar `lsmod` nuevamente, se podrá apreciar el módulo `simple` como parte del enlistado:

```
alex@alex-VirtualBox:~$ sudo insmod simple.ko
[sudo] password for alex:
alex@alex-VirtualBox:~$ lsmod
Module                Size  Used by
simple                  16384  0
edac_mce_amd          28672  0
crc10dif_pclmul       16384  1
crc32_pclmul          16384  0
ghash_clmulni_intel   16384  0
aesni_intel           372736  0
snd_intel8x0           45056  2
aes_x86_64            20480  1 aesni_intel
snd_ac97_codec         135168  1 snd_intel8x0
crypto_simd            16384  1 aesni_intel
cryptd                 24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
ac97_bus               16384  1 snd_ac97_codec
snd_pcm                102400  2 snd_intel8x0,snd_ac97_codec
glue_helper            16384  1 aesni_intel
vmwgfx                290816  2
snd_seq_midi           20480  0
joydev                 28672  0
snd_seq_midi_event     16384  1 snd_seq_midi
ttm                    102400  1 vmwgfx
snd_rawmidi            36864  1 snd_seq_midi
drm_kms_helper         180224  1 vmwgfx
snd_seq                69632  2 snd_seq_midi,snd_seq_midi_event
drm                    479232  5 vmwgfx,drm_kms_helper,ttm
fb_sys_fops            16384  1 drm_kms_helper
```

Figura 8. Adición del módulo `simple` a la lista de módulos cargados al kernel.

Sin embargo, uno recordará que el módulo está configurado de tal manera que, al ser cargado al kernel, se produciría un mensaje informativo. Para leer tal mensaje, se invoca el comando `dmesg`, el cual imprime el buffer de mensajes del kernel:


```
alex@alex-VirtualBox:~$ dmesg
[ 0.000000] Linux version 5.0.0-23-generic (build@lgw01-amd64-030) (gcc ver
sion 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #24~18.04.1-Ubuntu SMP Mon Jul 29 1
6:12:28 UTC 2019 (Ubuntu 5.0.0-23.24~18.04.1-generic 5.0.15)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.0.0-23-generic root=UI
D=8e96d381-57c2-49af-b1a1-d6cff9b379b5 ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Hygon HygonGenuine
[ 0.000000] Centaur CentaurHauls
[ 0.000000] [Firmware Bug]: TSC doesn't count with P0 frequency!
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
using 'standard' format.
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbff] usable
[ 0.000000] BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000000f0000-0x000000000000fffff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000000100000-0x000000000003ffeffff] usable
[ 0.000000] BIOS-e820: [mem 0x000000000003fff0000-0x000000000003ffffff] ACPI data
[ 0.000000] BIOS-e820: [mem 0x000000000fec00000-0x000000000fec00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000fee00000-0x000000000fee00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000fffc0000-0x000000000ffffff] reserved
[ 0.000000] NX (Execute Disable) protection: active
```

```
ration="profile_load" profile="unconfined" name="snap.gnome-calculator.gnome-ca
lculator" pid=526 comm="apparmor_parser"
[ 47.164584] audit: type=1400 audit(1583296543.196:26): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-characters.gnome-ch
aracters" pid=527 comm="apparmor_parser"
[ 47.361492] audit: type=1400 audit(1583296543.392:27): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-logs.gnome-logs" pi
d=528 comm="apparmor_parser"
[ 47.504338] audit: type=1400 audit(1583296543.532:28): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-system-monitor.gnom
e-system-monitor" pid=535 comm="apparmor_parser"
[ 47.583709] audit: type=1400 audit(1583296543.612:29): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.core" pid=546 c
omm="apparmor_parser"
[ 47.690592] audit: type=1400 audit(1583296543.720:30): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.gnome-calculato
r" pid=550 comm="apparmor_parser"
[ 47.882647] audit: type=1400 audit(1583296543.912:31): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.gnome-character
s" pid=551 comm="apparmor_parser"
[ 57.493132] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control
: RX
[ 57.506781] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[ 114.206731] rfkill: input handler disabled
[ 2467.456194] simple: loading out-of-tree module taints kernel.
[ 2467.456300] simple: module verification failed: signature and/or required ke
y missing - tainting kernel
[ 2467.458700] Loading Module
```

Figuras 9, 10. Salidas de todo el buffer de mensajes del kernel. Mayormente mensajes producidos

por manejadores de dispositivos. Hasta el final, el mensaje correspondiente del módulo `simple.c`

En caso de querer remover un módulo del kernel, se empleará el comando `rmmod`.

Para verificar la exitosa eliminación del módulo, nuevamente se invoca a `dmesg`:

```
alex@alex-VirtualBox:~$ sudo rmmod simple.ko
alex@alex-VirtualBox:~$ dmesg
[ 0.000000] Linux version 5.0.0-23-generic (buildd@lgw01-amd64-030) (gcc ver
sion 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #24~18.04.1-Ubuntu SMP Mon Jul 29 1
6:12:28 UTC 2019 (Ubuntu 5.0.0-23.24~18.04.1-generic 5.0.15)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.0.0-23-generic root=UUI
D=8e96d381-57c2-49af-b1a1-d6cff9b379b5 ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Hygon HygonGenuine
[ 0.000000] Centaur CentaurHauls
[ 0.000000] [Firmware Bug]: TSC doesn't count with P0 frequency!
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
using 'standard' format.
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbfff] usable
[ 0.000000] BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved
[ 0.000000] BIOS-e820: [mem 0x000000000000f0000-0x000000000000fffff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000000100000-0x000000000003fffff] usable
[ 0.000000] BIOS-e820: [mem 0x000000000003fff000-0x000000000003fffff] ACPI data
[ 0.000000] BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffffff] reserved
```

```
lculator" pid=526 comm="apparmor_parser"
[ 47.164584] audit: type=1400 audit(1583296543.196:26): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-characters.gnome-ch
aracters" pid=527 comm="apparmor_parser"
[ 47.361492] audit: type=1400 audit(1583296543.392:27): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-logs.gnome-logs" pi
d=528 comm="apparmor_parser"
[ 47.504338] audit: type=1400 audit(1583296543.532:28): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap.gnome-system-monitor.gnom
e-system-monitor" pid=535 comm="apparmor_parser"
[ 47.583709] audit: type=1400 audit(1583296543.612:29): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.core" pid=546 c
omm="apparmor_parser"
[ 47.690592] audit: type=1400 audit(1583296543.720:30): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.gnome-calculato
r" pid=550 comm="apparmor_parser"
[ 47.882647] audit: type=1400 audit(1583296543.912:31): apparmor="STATUS" ope
ration="profile_load" profile="unconfined" name="snap-update-ns.gnome-character
s" pid=551 comm="apparmor_parser"
[ 57.493132] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control
: RX
[ 57.506781] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[ 114.206731] rfkill: input handler disabled
[ 2467.456194] simple: loading out-of-tree module taints kernel.
[ 2467.456300] simple: module verification failed: signature and/or required ke
y missing - tainting kernel
[ 2467.458700] Loading Module
[ 2749.893679] Removing Module
```


Figuras 11, 12. Módulo `simple.c` removido del kernel exitosamente, junto a mensaje correspondiente impreso.

Parte II

En esta parte, se modificará el archivo “simple.c” y se creará una estructura “student” para que pueda crear 5 estudiantes

```

root@kali:~# cat simple.c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/list.h>
#include <linux/types.h>
#include <linux/slab.h>

struct student{
    int id;
    int age;
    int semester;
    char name[10];
    struct list_head list;
};

static LIST_HEAD(student_list);

int simple_init(void){
    printk(KERN_INFO "Loading the module!\n");

    struct student *person1;
    person1 = kmalloc(sizeof(*person1), GFP_KERNEL);
    person1 → id = 160805;
    person1 → age = 22;
    person1 → semester = 6;
    strcpy(person1→name, "Rafael");
    INIT_LIST_HEAD(&person1→list);
    list_add_tail(&person1→list, &student_list);

    struct student *person2;
    person2 = kmalloc(sizeof(*person2), GFP_KERNEL);
    person2 → id = 162819;
    person2 → age = 20;
    person2 → semester = 4;
    strcpy(person2→name, "Valeria");
    INIT_LIST_HEAD(&person2→list);
    list_add_tail(&person2→list, &student_list);

    struct student *person3;
    person3 = kmalloc(sizeof(*person3), GFP_KERNEL);
    person3 → id = 160046;
    person3 → age = 20;
    person3 → semester = 6;
    strcpy(person3→name, "Alexander");
    INIT_LIST_HEAD(&person3→list);
    list_add_tail(&person3→list, &student_list);

    struct student *person4;
    person4 = kmalloc(sizeof(*person4), GFP_KERNEL);
    person4 → id = 158320;
    person4 → age = 20;
    person4 → semester = 6;
    strcpy(person4→name, "Ashamat");
    INIT_LIST_HEAD(&person4→list);
    list_add_tail(&person4→list, &student_list);

    struct student *person5;
    person5 = kmalloc(sizeof(*person5), GFP_KERNEL);

```

```

person4 → age = 20;
person4 → semester = 6;
strcpy(person4→name, "Ashamat");
INIT_LIST_HEAD(&person4→list);
list_add_tail(&person4→list, &student_list);

struct student *person5;
person5 = kmalloc(sizeof(*person5), GFP_KERNEL);
person5 → id = 159381;
person5 → age = 20;
person5 → semester = 6;
strcpy(person5→name, "Ivan");
INIT_LIST_HEAD(&person5→list);
list_add_tail(&person5→list, &student_list);

printk(KERN_INFO "Student list");
struct student *sPtr;

list_for_each_entry(sPtr, &student_list, list){
    printk(KERN_INFO "Student:\nID: %i\tAge: %i\tSemester: %i\tName: %s\n",
        sPtr→id,
        sPtr→age,
        sPtr→semester,
        sPtr→name
    );
}

return 0;
}

void simple_exit(void){
printk(KERN_INFO "Removing module! \n");

struct student *sPtr, *next;

list_for_each_entry_safe(sPtr, next, &student_list, list) {
    printk(KERN_INFO "Removing Student:\nID: %i\tAge: %i\tSemester: %i\tName: %s\n",
        sPtr→id,
        sPtr→age,
        sPtr→semester,
        sPtr→name
    );

    list_del(&sPtr→list);
    kfree(sPtr);
}

}

module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("UDLAP");
MODULE_DESCRIPTION("Simple Module - Append");
MODULE_AUTHOR("SOUL RELEASE");
root@kali:~#

```


En la segunda parte, el programa requiere de igual forma que en la primera parte el archivo MAKEFILE para poder compilar nuestro programa como se muestra en la siguiente imagen:

```
donrafiki@donrafiki-VirtualBox: ~/Documents/OS
File Edit View Search Terminal Help
donrafiki@donrafiki-VirtualBox:~/Documents/OS$ make
make -C /lib/modules/5.3.0-40-generic/build M=/home/donrafiki/Documents/OS modules
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-40-generic'
CC [M] /home/donrafiki/Documents/OS/simple.o
/home/donrafiki/Documents/OS/simple.c: In function 'simple_init':
/home/donrafiki/Documents/OS/simple.c:21:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *person1;
^
/home/donrafiki/Documents/OS/simple.c:30:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *person2;
^
/home/donrafiki/Documents/OS/simple.c:39:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *person3;
^
/home/donrafiki/Documents/OS/simple.c:48:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *person4;
^
/home/donrafiki/Documents/OS/simple.c:57:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *person5;
^
/home/donrafiki/Documents/OS/simple.c:67:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *sPtr;
^
/home/donrafiki/Documents/OS/simple.c: In function 'simple_exit':
/home/donrafiki/Documents/OS/simple.c:83:1: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct student *sPtr, *next;
^
Building modules, stage 2.
MODPOST 1 modules
CC /home/donrafiki/Documents/OS/simple.mod.o
LD [M] /home/donrafiki/Documents/OS/simple.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-40-generic'
donrafiki@donrafiki-VirtualBox:~/Documents/OS$
```

Al igual que en la primera parte, es esencial agregarlo al kernel.

```
donrafiki@donrafiki-VirtualBox: ~/Documents/OS
File Edit View Search Terminal Help
donrafiki@donrafiki-VirtualBox:~/Documents/OS$ ls -la
total 120
drwxr-xr-x 2 donrafiki donrafiki 4096 mar  4 20:27 .
drwxr-xr-x 3 donrafiki donrafiki 4096 mar  2 20:34 ..
-rw-r--r-- 1 donrafiki donrafiki 154 mar  2 21:30 Makefile
-rw-r--r-- 1 donrafiki donrafiki 39 mar  4 20:27 modules.order
-rw-r--r-- 1 donrafiki donrafiki 0 mar  4 10:17 Module.symvers
-rw-r--r-- 1 donrafiki donrafiki 2391 mar  4 20:26 simple.c
-rw-r--r-- 1 donrafiki donrafiki 6080 mar  4 20:27 simple.ko
-rw-r--r-- 1 donrafiki donrafiki 266 mar  4 20:27 .simple.ko.cmd
-rw-r--r-- 1 donrafiki donrafiki 39 mar  4 20:27 simple.mod
-rw-r--r-- 1 donrafiki donrafiki 646 mar  4 20:27 simple.mod.c
-rw-r--r-- 1 donrafiki donrafiki 144 mar  4 20:27 .simple.mod.cmd
-rw-r--r-- 1 donrafiki donrafiki 2800 mar  4 20:27 simple.mod.o
-rw-r--r-- 1 donrafiki donrafiki 30749 mar  4 20:27 .simple.mod.o.cmd
-rw-r--r-- 1 donrafiki donrafiki 4120 mar  4 20:27 simple.o
-rw-r--r-- 1 donrafiki donrafiki 30996 mar  4 20:27 .simple.o.cmd
donrafiki@donrafiki-VirtualBox:~/Documents/OS$
```

Nuestro programa crea una estructura de nombre "Student" donde se guardan los datos:

- Id
- Edad
- Año
- Nombre

Para este ejemplo se guardaron 5 datos. Se comprueba que funcionó con el comando dmesg().

```
donrafiki@donrafiki-VirtualBox: ~/Documents/OS
File Edit View Search Terminal Help
donrafiki@donrafiki-VirtualBox:~/Documents/OS$ sudo insmod simple.ko
donrafiki@donrafiki-VirtualBox:~/Documents/OS$ dmesg
[ 1588.925239] Loading the module!
[ 1588.925241] Student list
[ 1588.925242] Student:
ID: 160805      Age: 22 Semester: 6      Name: Rafael
[ 1588.925243] Student:
ID: 162819      Age: 20 Semester: 4      Name: Valeria
[ 1588.925244] Student:
ID: 160046      Age: 20 Semester: 6      Name: Alexander
[ 1588.925245] Student:
ID: 158320      Age: 20 Semester: 6      Name: Ashamat
[ 1588.925245] Student:
ID: 159381      Age: 20 Semester: 6      Name: Ivan
donrafiki@donrafiki-VirtualBox:~/Documents/OS$
```

Como se logra observar, el programa funciona correctamente mostrando los 5 datos de tipo estructura.

Al momento de eliminar el módulo se elimina cada uno de los elementos en el.

La funcion kfree() devuelve la memoria al núcleo.

Conclusiones

Durante el desarrollo de esta práctica se aprendió la forma de crear, modificar y observar los comportamientos de los distintos módulos del kernel Linux. Para poder acceder a ellos es necesario declararlos e importar las librerías correspondientes del kernel y los módulos de Linux.

Bibliografía

Silberschatz, A., Galvin, P. B., & Gagne, G. (9th Ed.). (2013). *Operating System Concepts*. Wiley.

[SolidusCode]. (2012, 12 de mayo). *Linux Kernel Module Programming - 03 Coding, Compiling the Module* [Video]. YouTube. <https://www.youtube.com/watch?v=S8hiflrDh-g>

The Linux Documentation Project. (2020). *The Linux Kernel Module Programming Guide* [PDF]. <https://www.tldp.org/LDP/lkmpg/2.4/lkmpg.pdf>

Unix Makefile Tutorial. (2020). *tutorialspoint*. <https://www.tutorialspoint.com/makefile/index.htm>