

Mini-project: Adding A Kernel Module

Aim:

To add a kernel module to an existing base kernel.

Theory:

What is a Kernel?

The kernel is the central module of an operating system (OS). It is the part of the operating system that loads first, and it remains in main memory. Because it stays in memory, it is important for the kernel to be as small as possible while still providing all the essential services required by other parts of the operating system and applications. The kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.

Typically, the kernel is responsible for memory management, process and task management, and disk management. The kernel connects the system hardware to the application software. Every operating system has a kernel. For example, the Linux kernel is used by numerous operating systems including Linux, FreeBSD, Android, and others.

What is a Kernel Module?

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.

To create a kernel module, you can read The Linux Kernel Module Programming Guide. A module can be configured as built-in or loadable. To dynamically load or remove a module, it has to be configured as a loadable module in the kernel configuration (the line related to the module will, therefore, display the letter M).

The kernel modules will have a .ko extension. On a normal Linux system, the kernel modules will reside inside `/lib/modules/<kernel_version>/kernel/` directory.

Some Important Commands

Modules are stored in `/usr/lib/modules/kernel_release`. You can use the command `uname -r` to get your current kernel release version.

To show what kernel modules are currently loaded:

```
$ lsmod
```

To show information about a module:

```
$ modinfo module_name
```

To list the options that are set for a loaded module:

```
$ systool -v -m module_name
```

To display the comprehensive configuration of all the modules:

```
$ modprobe -c | less
```

To display the configuration of a particular module:

```
$ modprobe -c | grep module_name
```

List the dependencies of a module (or alias), including the module itself:

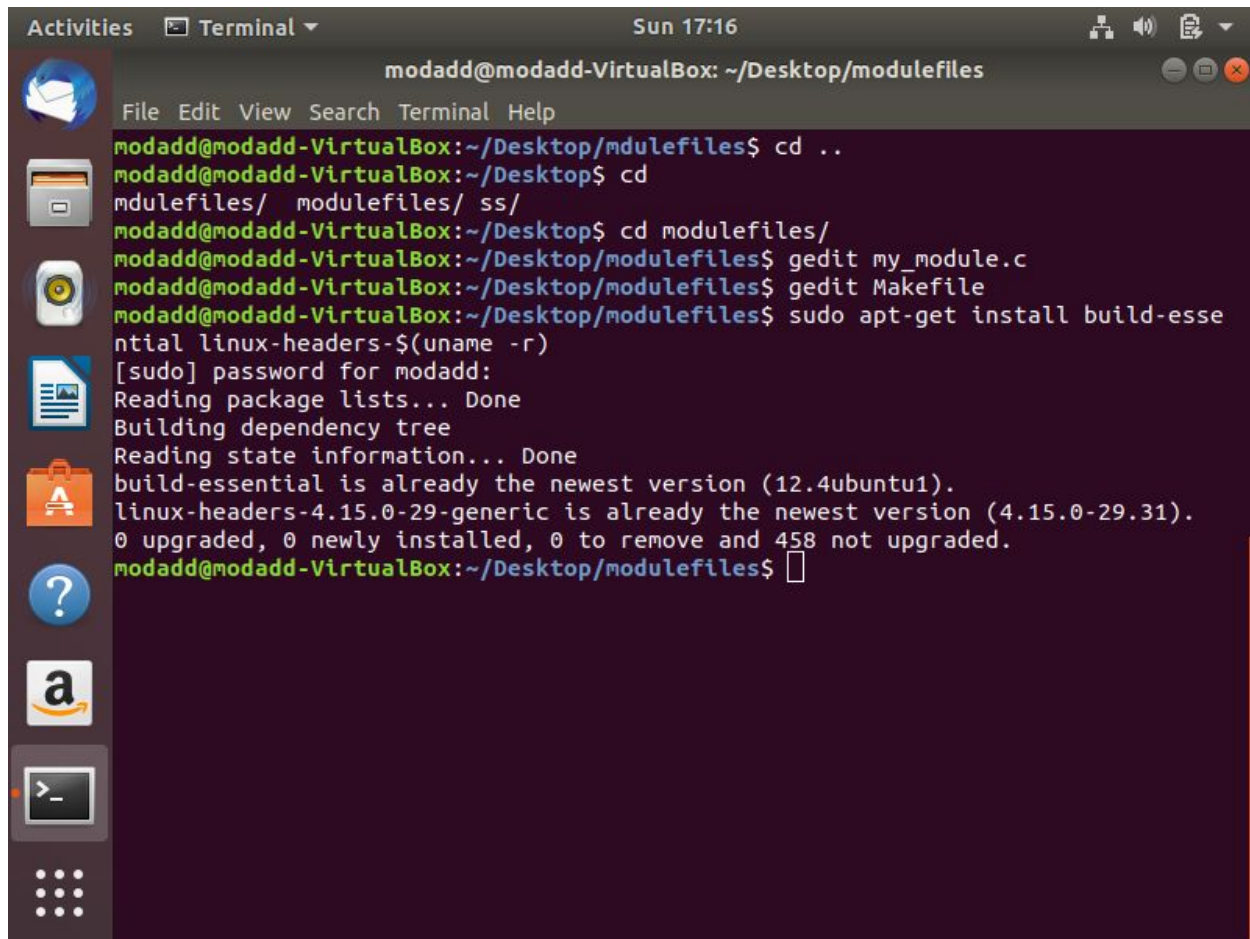
```
$ modprobe --show-depends module_name
```

*The above Commands will be used to step-by-step while the module addition.

Procedure:

1. Installing the linux headers

```
# apt-get install build-essential linux-headers-$(uname -r)
```



The screenshot shows a terminal window titled "modadd@modadd-VirtualBox: ~/Desktop/modulefiles". The terminal output is as follows:

```
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ cd ..
modadd@modadd-VirtualBox:~/Desktop$ cd
modulefiles/ modulefiles/ ss/
modadd@modadd-VirtualBox:~/Desktop$ cd modulefiles/
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ gedit my_module.c
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ gedit Makefile
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ sudo apt-get install build-ess
ntial linux-headers-$(uname -r)
[sudo] password for modadd:
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
linux-headers-4.15.0-29-generic is already the newest version (4.15.0-29.31).
0 upgraded, 0 newly installed, 0 to remove and 458 not upgraded.
modadd@modadd-VirtualBox:~/Desktop/modulefiles$
```

2. Kernel Module Source Code

```
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_INFO
#include <linux/init.h>      // included for __init and __exit macros

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author");
MODULE_DESCRIPTION("Test module for addition");

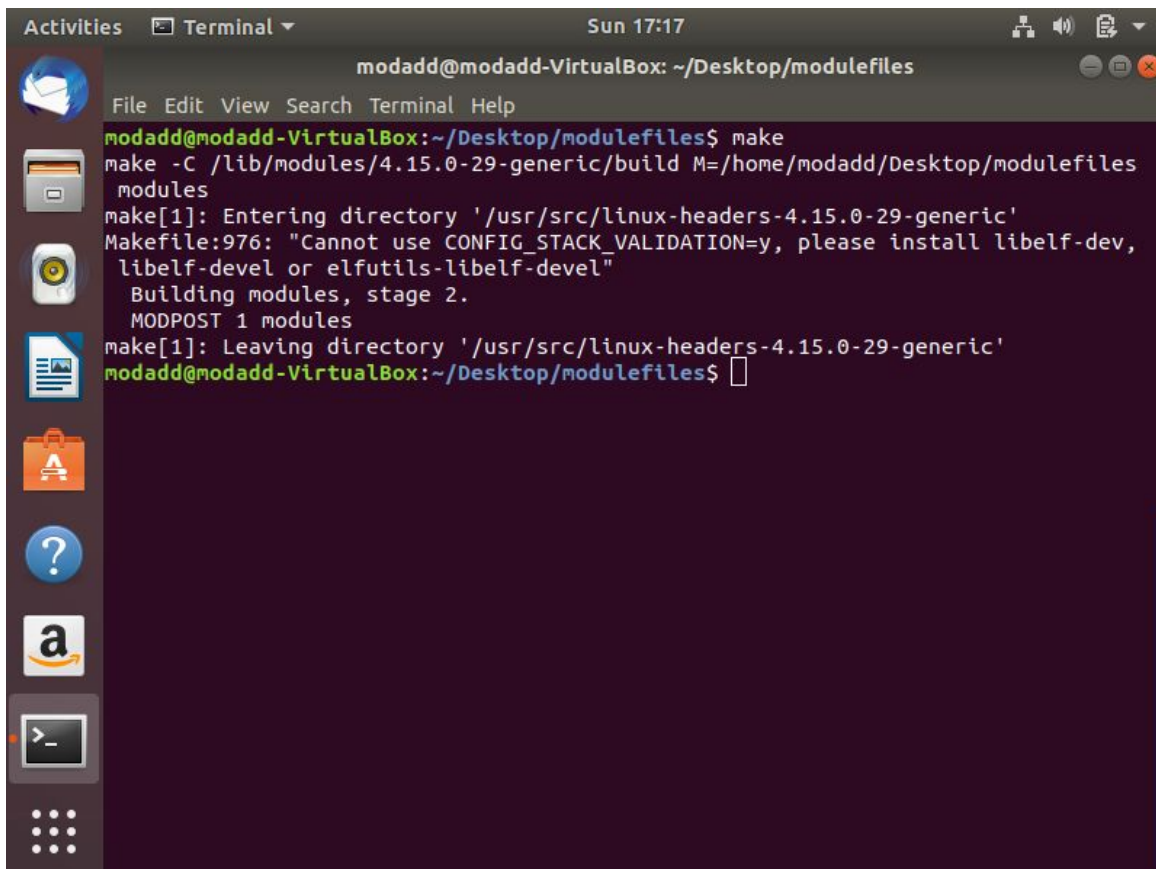
static int __init newmod_init(void) {
    printk(KERN_INFO "NEW MODULE IS INSERTED\n");
    return 0;    // Non-zero return means that the module couldn't be
loaded.
}
```

```
static void __exit newmod_cleanup(void) {  
    printk(KERN_INFO "MODULE REMOVED\n");  
}  
module_init(newmod_init);  
module_exit(newmod_cleanup);
```

3. Create Makefile to Compile Kernel Module

```
obj-m += my_module.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
```

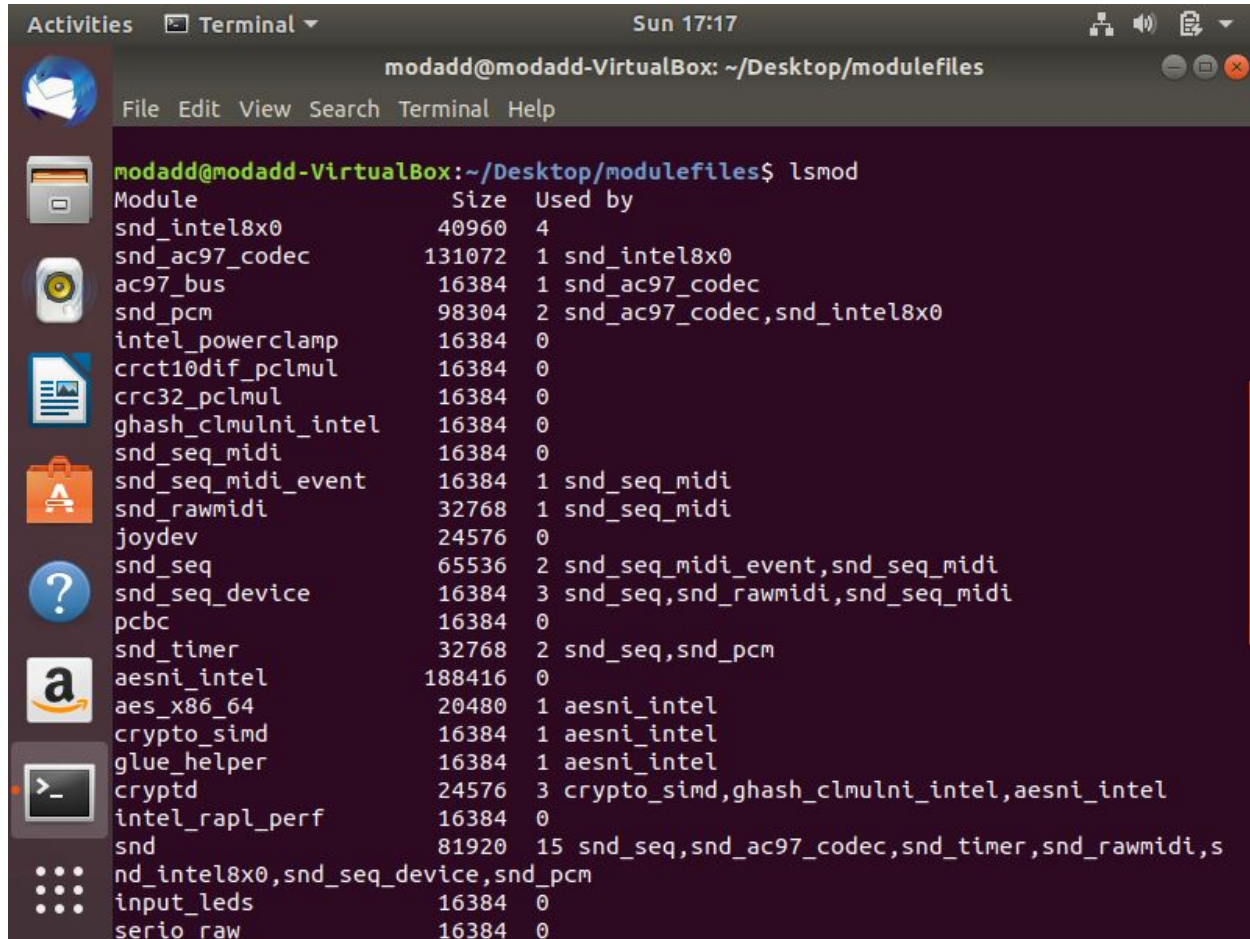
4. make command executed for module compilation and .ko file is created.



The screenshot shows a terminal window titled "modadd@modadd-VirtualBox: ~/Desktop/modulefiles". The terminal output is as follows:

```
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ make  
make -C /lib/modules/4.15.0-29-generic/build M=/home/modadd/Desktop/modulefiles  
modules  
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-29-generic'  
Makefile:976: "Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev,  
libelf-devel or elfutils-libelf-devel"  
Building modules, stage 2.  
MODPOST 1 modules  
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-29-generic'  
modadd@modadd-VirtualBox:~/Desktop/modulefiles$
```

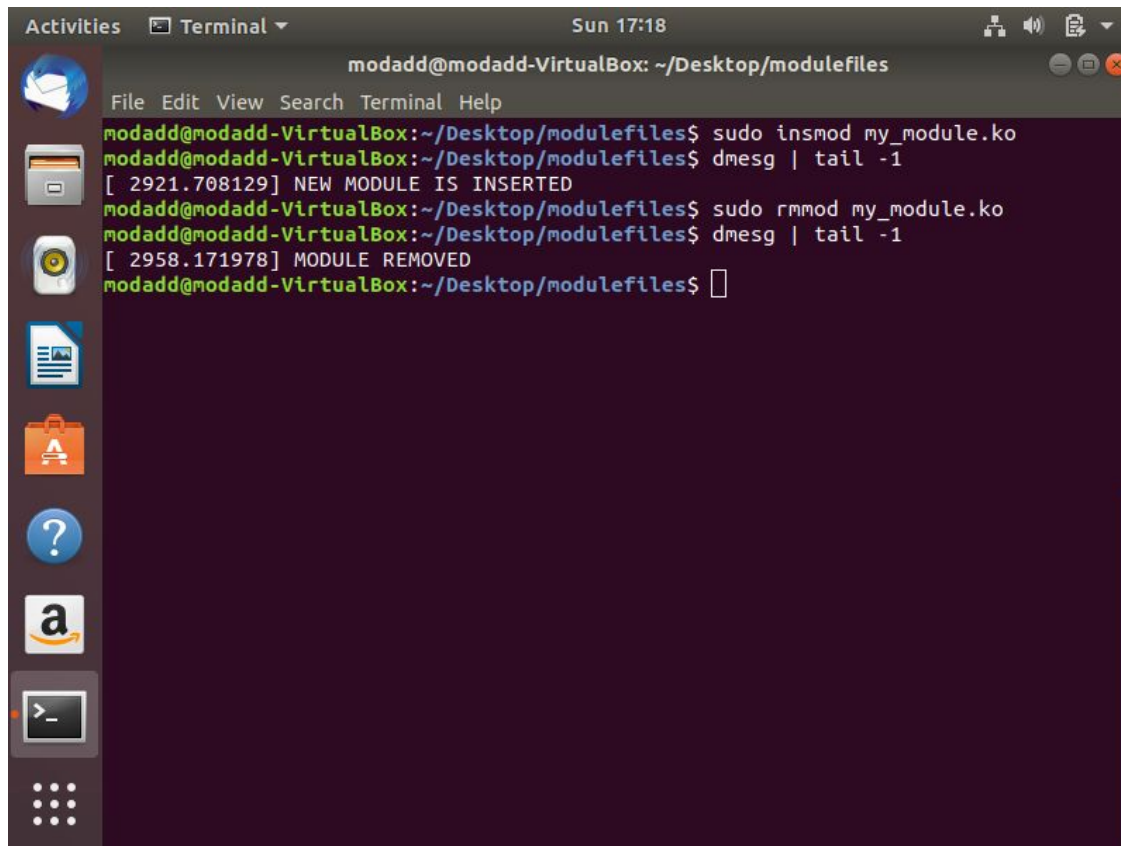
5. Lets see the current kernel modules



```
modadd@modadd-VirtualBox: ~/Desktop/modulefiles
File Edit View Search Terminal Help

modadd@modadd-VirtualBox:~/Desktop/modulefiles$ lsmod
Module                Size  Used by
snd_intel8x0           40960    4
snd_ac97_codec        131072    1 snd_intel8x0
ac97_bus               16384    1 snd_ac97_codec
snd_pcm               98304    2 snd_ac97_codec,snd_intel8x0
intel_powerclamp       16384    0
crct10dif_pclmul       16384    0
crc32_pclmul           16384    0
ghash_clmulni_intel    16384    0
snd_seq_midi           16384    0
snd_seq_midi_event     16384    1 snd_seq_midi
snd_rawmidi            32768    1 snd_seq_midi
joydev                 24576    0
snd_seq                65536    2 snd_seq_midi_event,snd_seq_midi
snd_seq_device         16384    3 snd_seq,snd_rawmidi,snd_seq_midi
pcbc                   16384    0
snd_timer              32768    2 snd_seq,snd_pcm
aesni_intel           188416    0
aes_x86_64             20480    1 aesni_intel
crypto_simd            16384    1 aesni_intel
glue_helper            16384    1 aesni_intel
cryptd                 24576    3 crypto_simd,ghash_clmulni_intel,aesni_intel
intel_rapl_perf        16384    0
snd                     81920   15 snd_seq,snd_ac97_codec,snd_timer,snd_rawmidi,s
nd_intel8x0,snd_seq_device,snd_pcm
input_leds             16384    0
serio_raw              16384    0
```

6. Look at the initialization message stored in the kernel message buffer after insertion of module(insmod), also after the removal of the module(rmmod).



```
modadd@modadd-VirtualBox: ~/Desktop/modulefiles
File Edit View Search Terminal Help
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ sudo insmod my_module.ko
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ dmesg | tail -1
[ 2921.708129] NEW MODULE IS INSERTED
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ sudo rmmod my_module.ko
modadd@modadd-VirtualBox:~/Desktop/modulefiles$ dmesg | tail -1
[ 2958.171978] MODULE REMOVED
modadd@modadd-VirtualBox:~/Desktop/modulefiles$
```

Conclusion:

A kernel object file is created when make file is compiled and a kernel object file is generated. This is the module that is inserted using the insmod command and the init macro is invoked and a message is added to the buffer similarly during rmmod; exit macro is executed and the message is added to the kernel buffer.

Thus, following the above concepts, I added a Kernel module to the existing OS module.