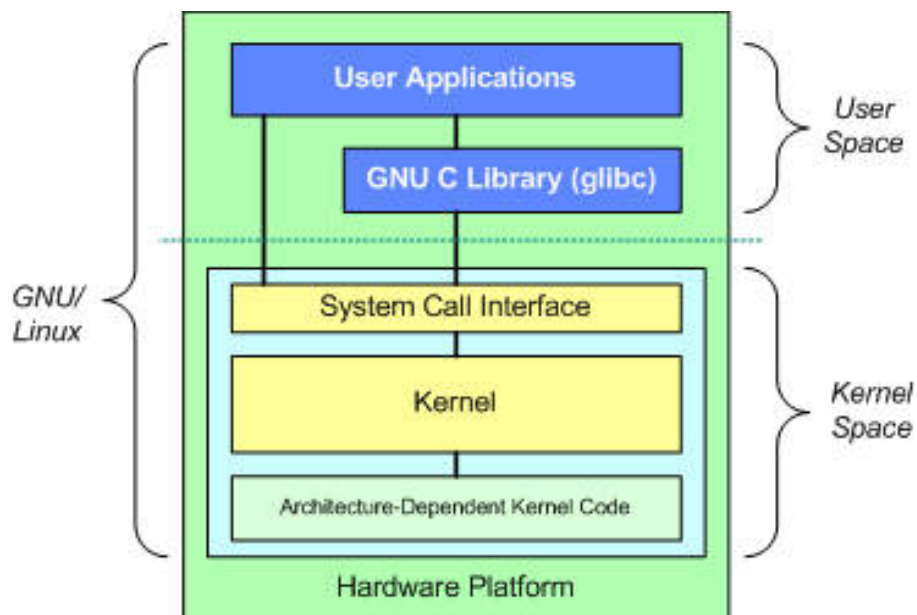


Introduction to Linux Kernel



- In linux system , several concurrent processes perform different tasks.
- Each process asks for system resources , be it computing power, memory , network connectivity or some other resources.
- The kernel is the big chunk of executable code in charge of handling all such requests.
- Although the distinction between the different kernel tasks isn't always clearly marked , the kernel's role can be split into the following parts:

Process Management
Memory Management
File System
Device Control
Networking

Process Management

- Kernel is in charge of creating and destroying processes.
- Communication among different processes.
- Scheduling the process.

Memory Management

- Computer's Memory is a major Resource and the policy used to deal with it is a critical one for system performnace.
- Kernel builds up a virtual addressing space on top of the limited available physical memory resources.

File System

- Linux is heavily based in the file system concept, almost everything in unix can be treated as a file
- Kernel builds a structured file system on top of unstructured hardware, and the resulting file abstraction is heavily used throughout the whole system.

Device Driver

- Almost every system operation eventually maps to a physical device, with the exception of the processor, memory etc.
- The Kernel must embed in it a device driver for every peripheral on a system, from the harddrive to the keyboard.

Networking

- Networking must be managed by the operating system, because most network operations are not specific to a process; incoming packets are asynchronous events.
- Packets must be collected, identified, and dispatched before a process takes care of them. For example, windows media player playing songs, flash player videos etc.
- System is in charge of delivering data packets across program and network interface, and it must control the execution of programs according to their network activity.
- Additionally, all the routing and address resolution issues are implemented within kernel. For example, local host, intranet, domain name search based on hosts files, etc.

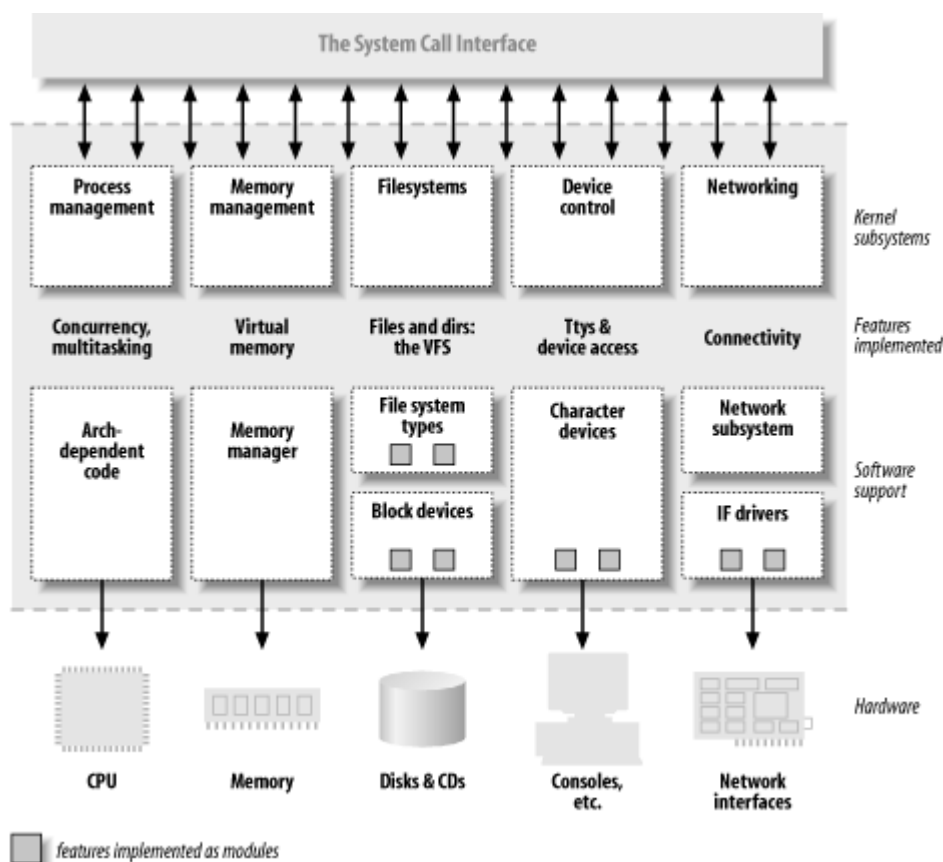
Introduction to Device Driver

- Device drivers take on a special role in the linux kernel.
- Device drivers completely hide the details of how the device works.
- User activities are performed by means of a set of standardized calls that are independent of the specified driver.
- Mapping those calls to device-specific operations that act on real hardware is then the role of the device driver.
- Drivers may be integrated directly into the kernel, or can be built separately from the rest of the kernel and 'plugged in' at runtime when needed, i.e. loadable modules.
- Each driver is different; as a driver writer, you need to understand your specific device well. But most of the principles and basic techniques are the same for all drivers.

Classes of Devices and Modules

Linux way of looking at devices distinguishes between three fundamental device types :

- Character Devices
- Block Devices
- Network Devices
- Linux Drivers/modules usually implement one of these types.
- This division of modules into different types, or classes , is not a rigid one. It is a good programming practice for scalability and extendibility.



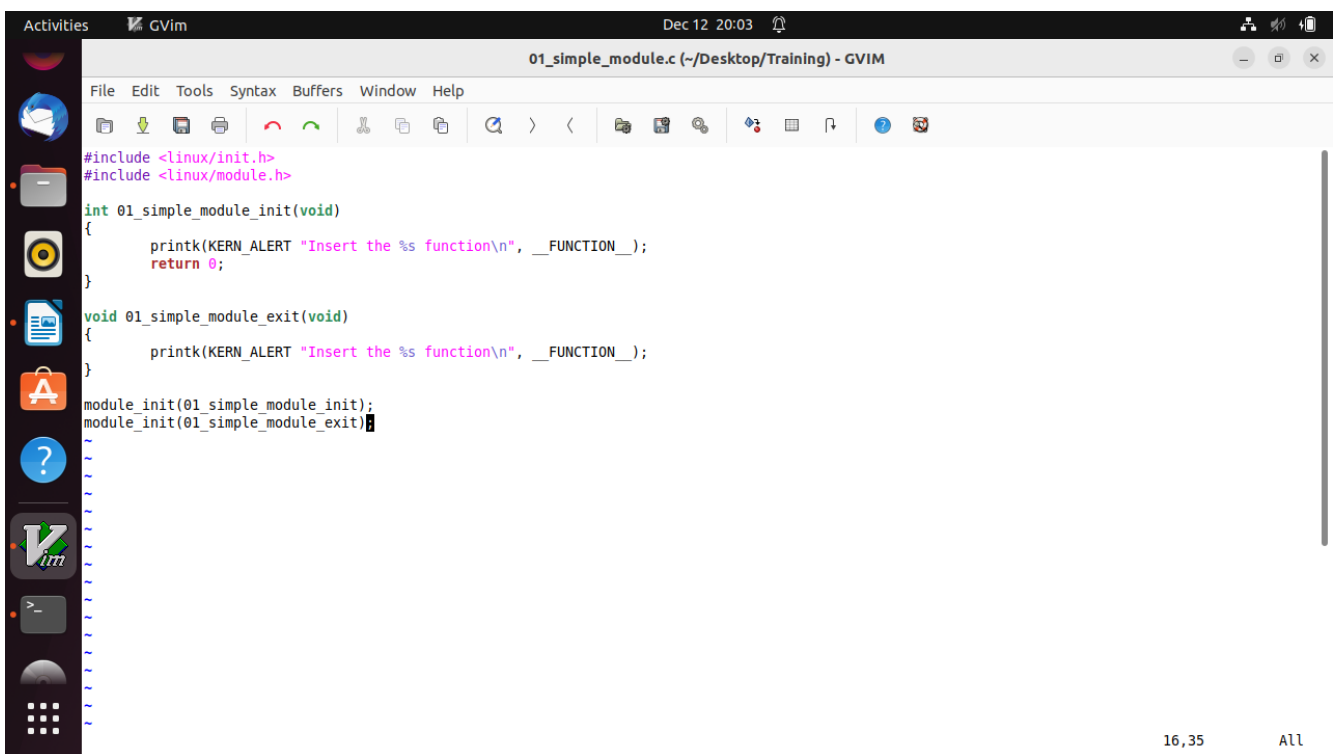
Character Drivers

- A character (char) device is one that can be accessed as a stream of bytes (like a file).
- A char driver is in charge of implementing this behaviour. For example , keyboard , mouse , camera , etc.
- The only relevant difference between a char device and a regular file is that you can always move back and forth in the regular file, whereas most char devices are just data channels , which you can only access sequentially.

Block Drivers

- Like char devices, block devices are accessed by file system nodes in the /dev directory.
- A block device is a device (e.g., a disk) that can host a file system. In most Unix systems, a block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes (or a larger power of two) bytes in length.
- Block device drivers permit random access.

Simple Loadable Kernel Module Example:-



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

int 01_simple_module_init(void)
{
    printk(KERN_ALERT "Insert the %s function\n", __FUNCTION__);
    return 0;
}

void 01_simple_module_exit(void)
{
    printk(KERN_ALERT "Insert the %s function\n", __FUNCTION__);
}

module_init(01_simple_module_init);
module_init(01_simple_module_exit);
```

Fig 1. Sample LKM .c file

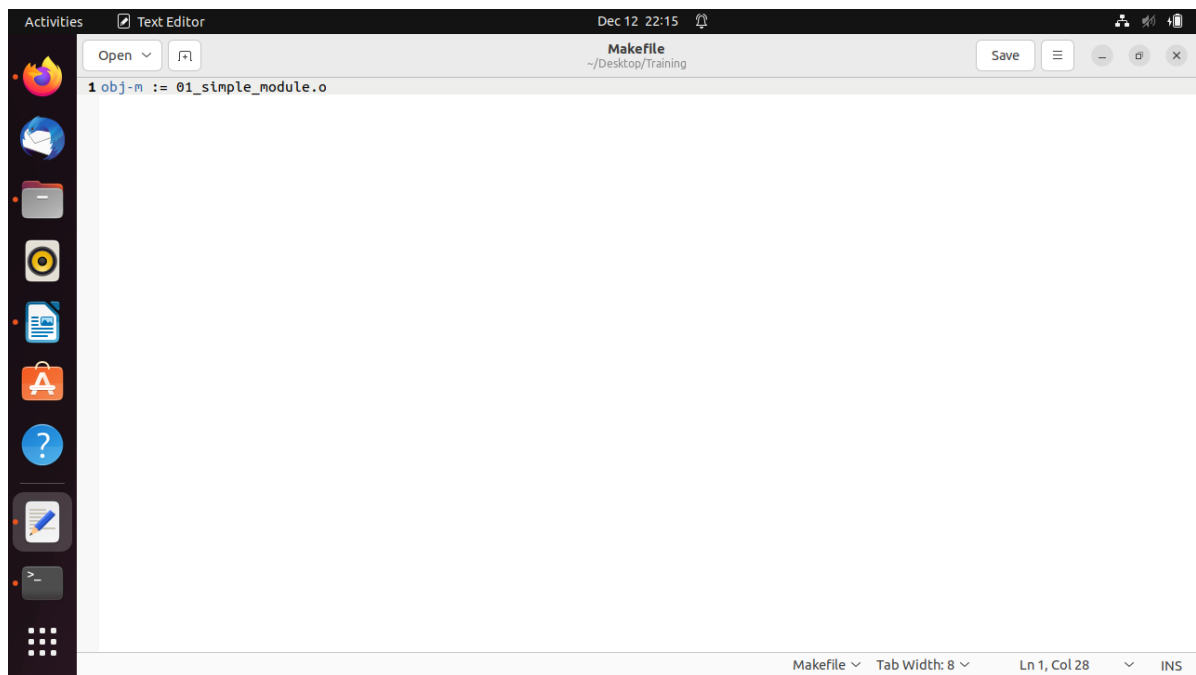


Fig 2. Makefile for the corresponding LKM .c file

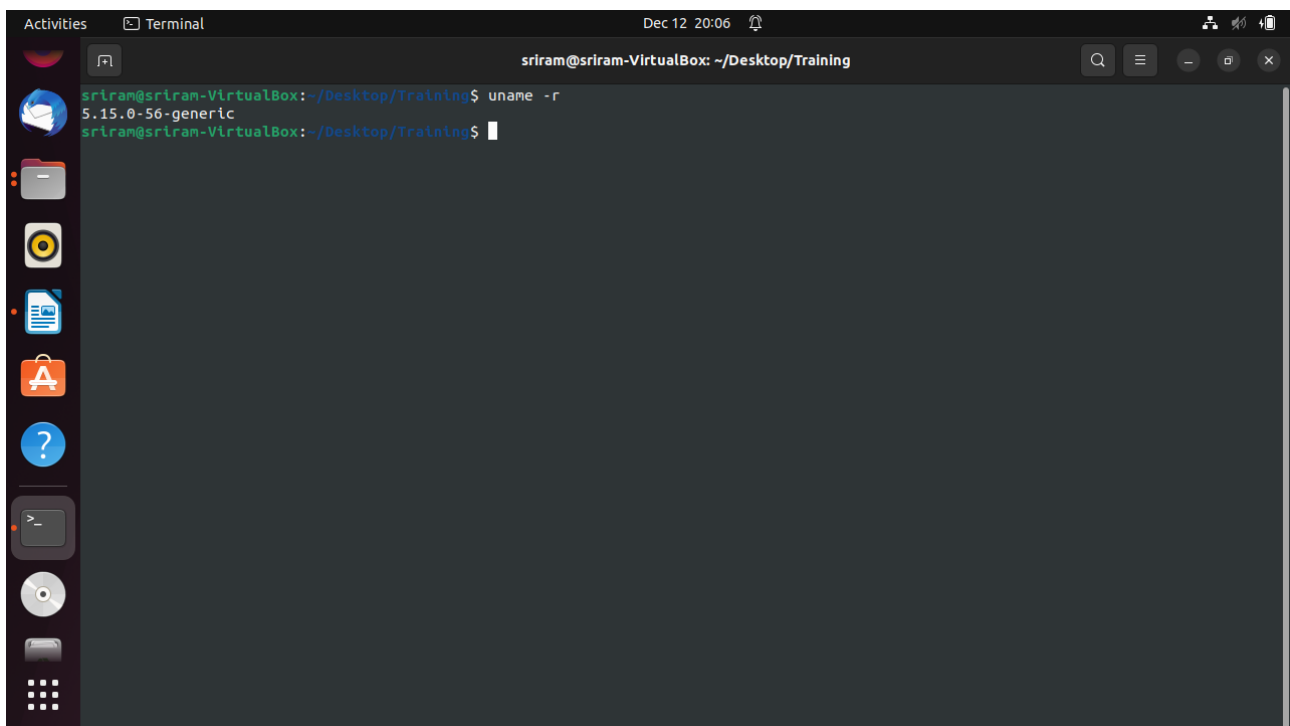


Fig 3. To display unix kernel version

```
Activities Terminal Dec 12 20:18 sriram@sriram-VirtualBox: ~/Desktop/Training

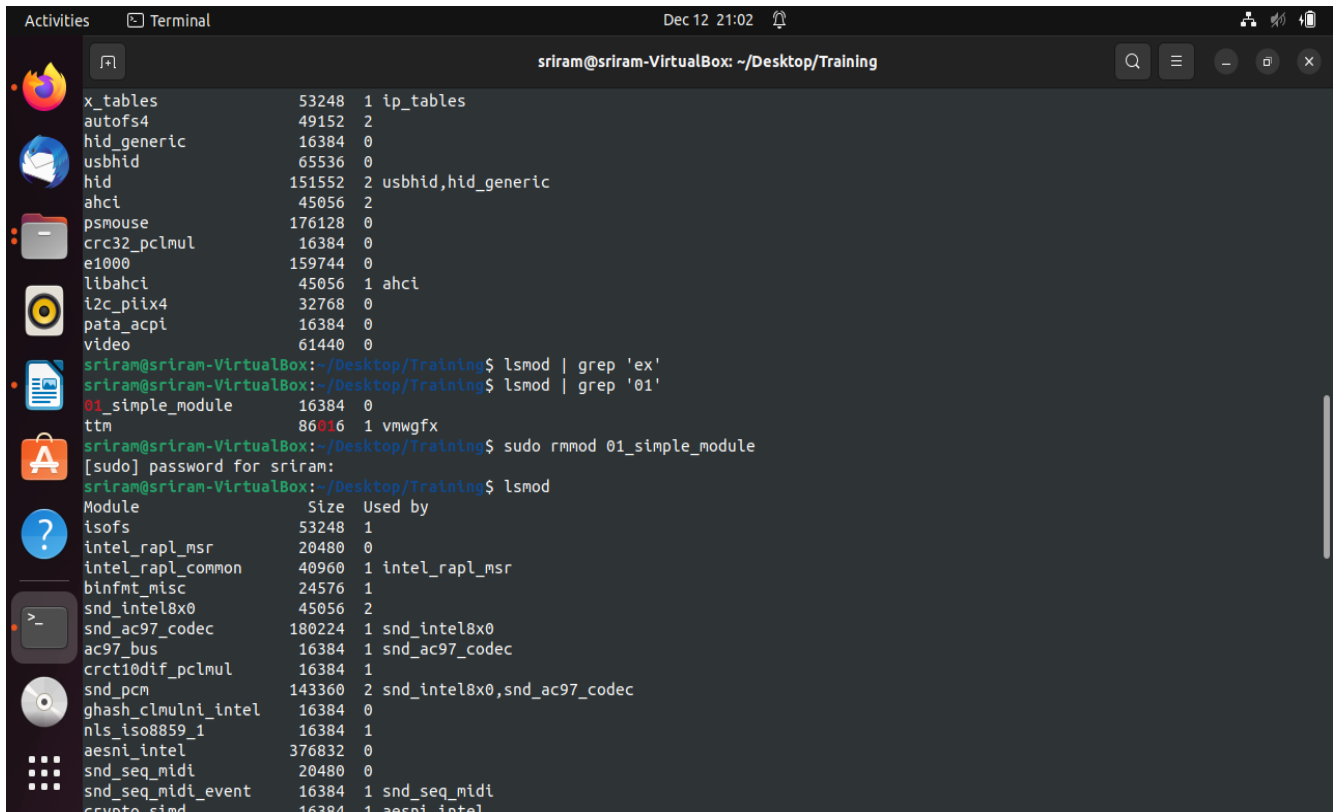
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$ make -C /lib/modules/$(uname -r)/build M=$PWD modules
make: Entering directory '/usr/src/linux-headers-5.15.0-56-generic'
CC [M] /home/sriram/Desktop/Training/01_simple_module.o
MODPOST /home/sriram/Desktop/Training/Module.symvers
CC [M] /home/sriram/Desktop/Training/01_simple_module.mod.o
LD [M] /home/sriram/Desktop/Training/01_simple_module.ko
BTF [M] /home/sriram/Desktop/Training/01_simple_module.ko
Skipping BTF generation for /home/sriram/Desktop/Training/01_simple_module.ko due to unavailability of vmlinux
make: Leaving directory '/usr/src/linux-headers-5.15.0-56-generic'
sriram@sriram-VirtualBox:~/Desktop/Training$ ls -l
total 152
-rw-rw-r-- 1 sriram sriram 365 Dec 12 20:18 01_simple_module.c
-rw-rw-r-- 1 sriram sriram 62376 Dec 12 20:18 01_simple_module.ko
-rw-rw-r-- 1 sriram sriram 50 Dec 12 20:18 01_simple_module.mod
-rw-rw-r-- 1 sriram sriram 855 Dec 12 20:18 01_simple_module.mod.c
-rw-rw-r-- 1 sriram sriram 50128 Dec 12 20:18 01_simple_module.mod.o
-rw-rw-r-- 1 sriram sriram 13640 Dec 12 20:18 01_simple_module.o
-rw-rw-r-- 1 sriram sriram 28 Dec 12 20:16 Makefile
-rw-rw-r-- 1 sriram sriram 50 Dec 12 20:18 modules.order
-rw-rw-r-- 1 sriram sriram 0 Dec 12 20:18 Module.symvers
sriram@sriram-VirtualBox:~/Desktop/Training$
```

Fig 4. Linux Command to create kernel Object and display in current working directory

```
Activities Terminal Dec 12 20:21 sriram@sriram-VirtualBox: ~/Desktop/Training

sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$
sriram@sriram-VirtualBox:~/Desktop/Training$ sudo insmod ./01_simple_module.ko
[sudo] password for sriram:
sriram@sriram-VirtualBox:~/Desktop/Training$ lsmod
Module                  Size  Used by
01_simple_module        16384  0
isofs                   53248  1
intel_rapl_msr          20480  0
intel_rapl_common       40960  1 intel_rapl_msr
binfmt_misc            24576  1
snd_intel8x0           45056  2
snd_ac97_codec         180224  1 snd_intel8x0
ac97_bus                16384  1 snd_ac97_codec
crct10dif_pclmul        16384  1
snd_pcm                143360  2 snd_intel8x0,snd_ac97_codec
ghash_c12mulni_intel    16384  0
nls_iso8859_1           16384  1
aesni_intel            376832  0
snd_seq_midi            20480  0
snd_seq_midi_event      16384  1 snd_seq_midi
crypto_simd             16384  1 aesni_intel
cryptd                  24576  2 crypto_simd,ghash_c12mulni_intel
snd_rawmidi             49152  1 snd_seq_midi
snd_seq                 77824  2 snd_seq_midi,snd_seq_midi_event
snd_seq_device          16384  3 snd_seq,snd_seq_midi,snd_rawmidi
snd_timer              40960  2 snd_seq,snd_pcm
joydev                  32768  0
input_leds              16384  0
snd                     106496  11 snd_seq,snd_seq_device,snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm,snd_rawmidi
vboxguest               45056  0
serio_raw               20480  0
soundcore               16384  1 snd
```

Fig 5. Command to display loaded modules in the kernel



A terminal window titled 'sriram@sriram-VirtualBox: ~/Desktop/Training' showing the following commands and output:

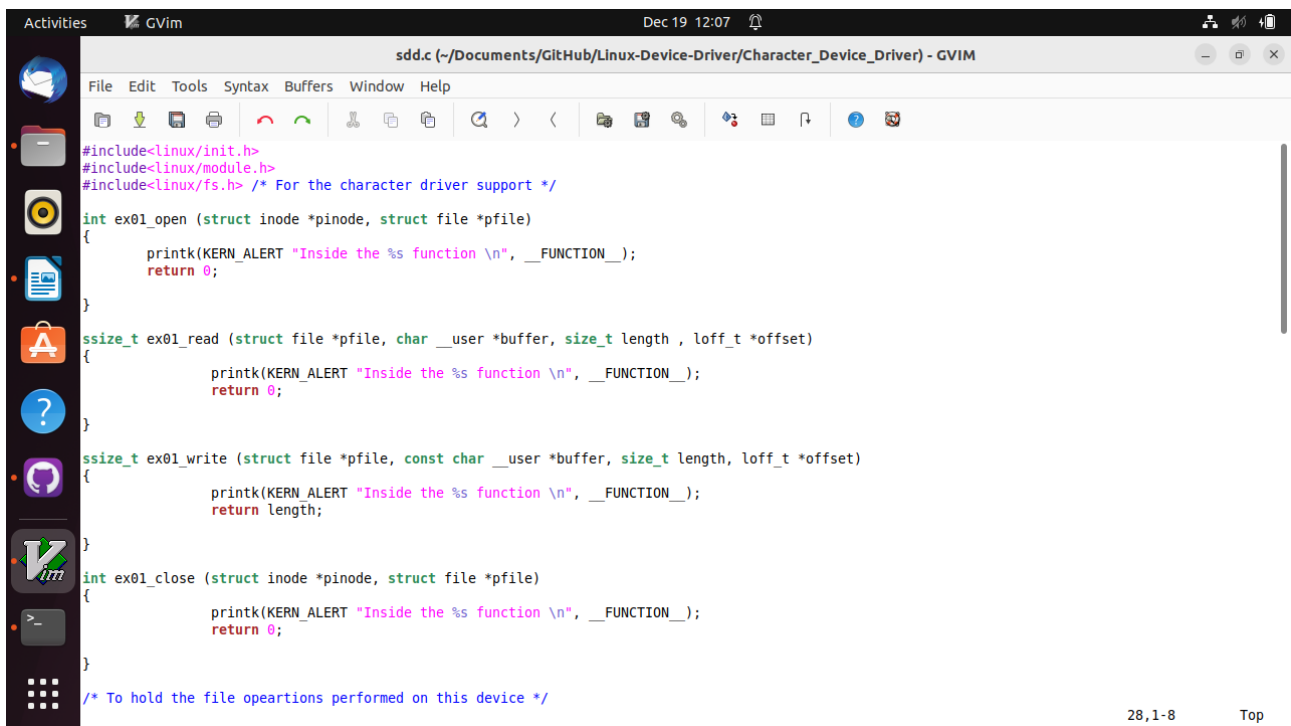
```
sriram@sriram-VirtualBox: ~/Desktop/Training$ lsmod | grep 'ex'
sriram@sriram-VirtualBox: ~/Desktop/Training$ lsmod | grep '01'
01_simple_module      16384  0
ttm                   86016  1 vmwgfx

sriram@sriram-VirtualBox: ~/Desktop/Training$ sudo rmmod 01_simple_module
[sudo] password for sriram:
sriram@sriram-VirtualBox: ~/Desktop/Training$ lsmod
```

Module	Size	Used by
isofs	53248	1
intel_rapl_msr	20480	0
intel_rapl_common	40960	1 intel_rapl_msr
binfmt_misc	24576	1
snd_intel8x0	45056	2
snd_ac97_codec	180224	1 snd_intel8x0
ac97_bus	16384	1 snd_ac97_codec
crct10dif_pclmul	16384	1
snd_pcm	143360	2 snd_intel8x0,snd_ac97_codec
ghash_clmulni_intel	16384	0
nls_iso8859_1	16384	1
aesni_intel	376832	0
snd_seq_midi	20480	0
snd_seq_midi_event	16384	1 snd_seq_midi
crypto_simd	16384	1 aesni_intel

Fig 5a. Command to display loaded modules in the kernel

Character Device Driver Example :



A GVIM editor window titled 'sdd.c (~/.Documents/GitHub/Linux-Device-Driver/Character_Device_Driver) - GVIM' showing the following C code:

```
#include<linux/init.h>
#include<linux/module.h>
#include<linux/fs.h> /* For the character driver support */

int ex01_open (struct inode *pinode, struct file *pfile)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);
    return 0;
}

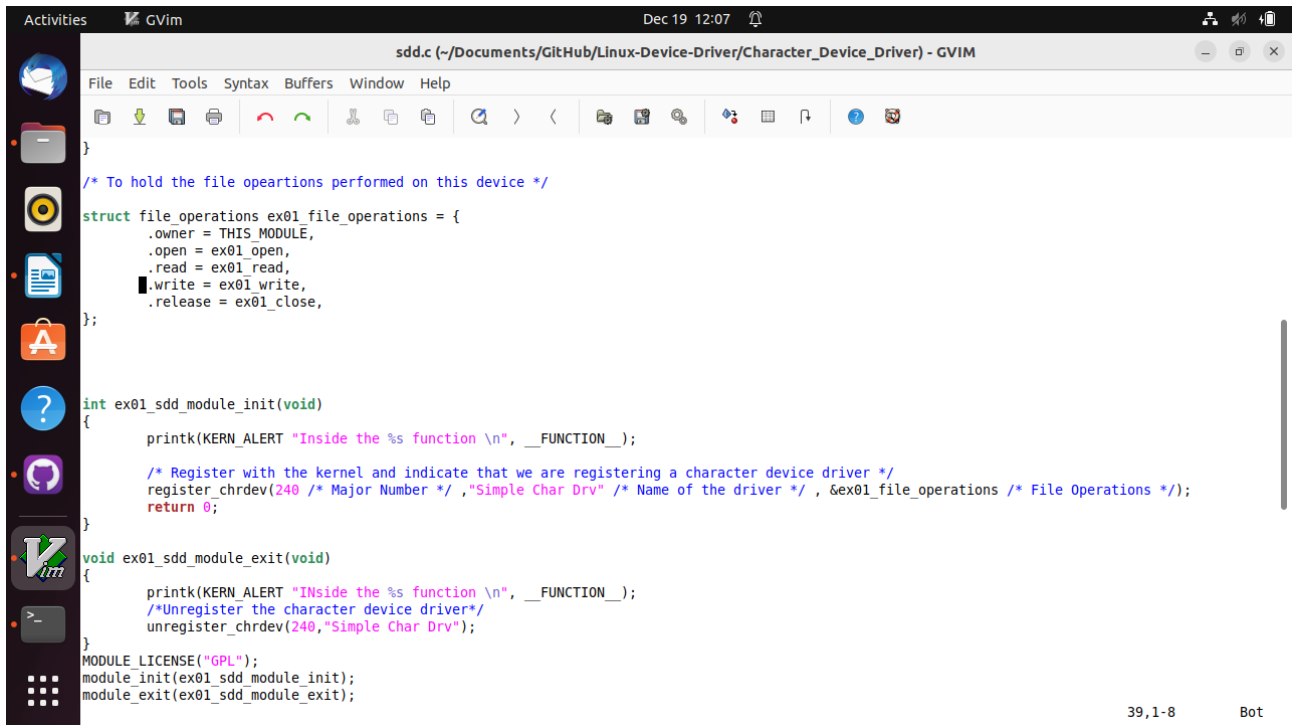
ssize_t ex01_read (struct file *pfile, char __user *buffer, size_t length , loff_t *offset)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);
    return 0;
}

ssize_t ex01_write (struct file *pfile, const char __user *buffer, size_t length, loff_t *offset)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);
    return length;
}

int ex01_close (struct inode *pinode, struct file *pfile)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);
    return 0;
}

/* To hold the file opeartions performed on this device */
```

Fig 1. Character Device Driver Code



```
sdd.c (~/.Documents/GitHub/Linux-Device-Driver/Character_Device_Driver) - GVIM
File Edit Tools Syntax Buffers Window Help
}
/* To hold the file opeartions performed on this device */
struct file_operations ex01_file_operations = {
    .owner = THIS_MODULE,
    .open = ex01_open,
    .read = ex01_read,
    .write = ex01_write,
    .release = ex01_close,
};

int ex01_sdd_module_init(void)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);

    /* Register with the kernel and indicate that we are registering a character device driver */
    register_chrdev(240 /* Major Number */, "Simple Char Drv" /* Name of the driver */, &ex01_file_operations /* File Operations */);
    return 0;
}

void ex01_sdd_module_exit(void)
{
    printk(KERN_ALERT "Inside the %s function \n", __FUNCTION__);
    /*Unregister the character device driver*/
    unregister_chrdev(240, "Simple Char Drv");
}

MODULE_LICENSE("GPL");
module_init(ex01_sdd_module_init);
module_exit(ex01_sdd_module_exit);
```

39,1-8 Bot

Fig 1a. Character Device Driver Code