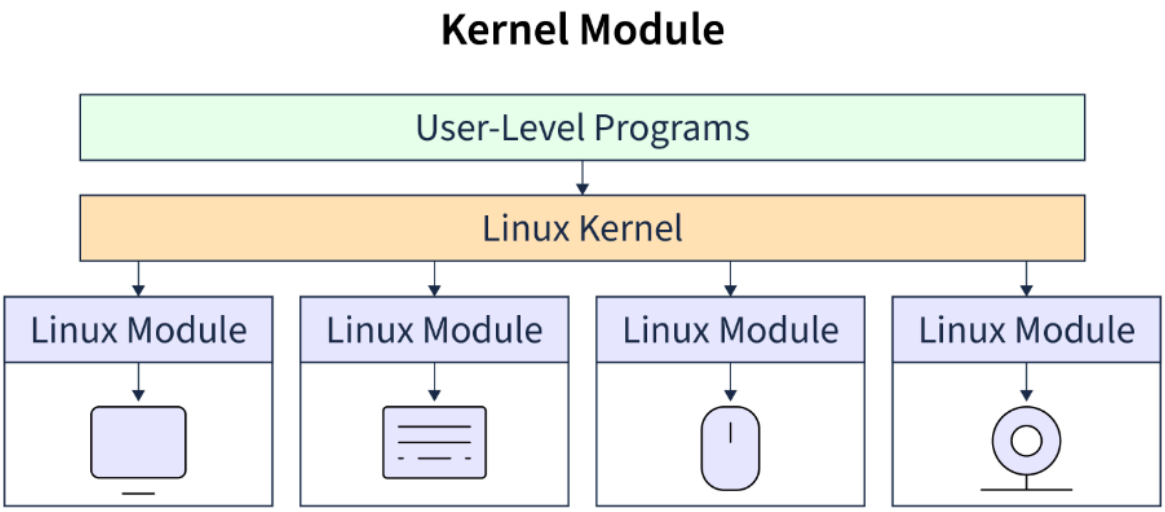
Name-Abhi Mehta

Reg No.-221080001

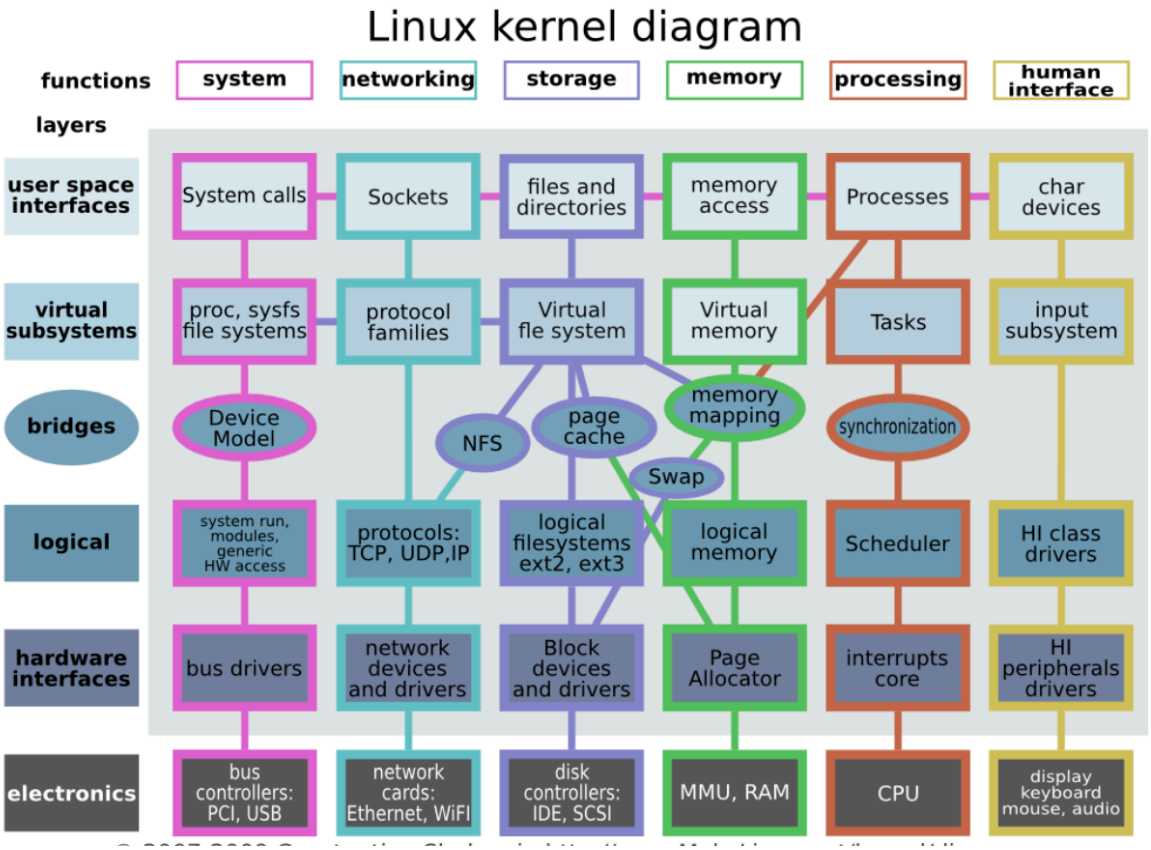
Experiment-10
Linux Kernel Modules

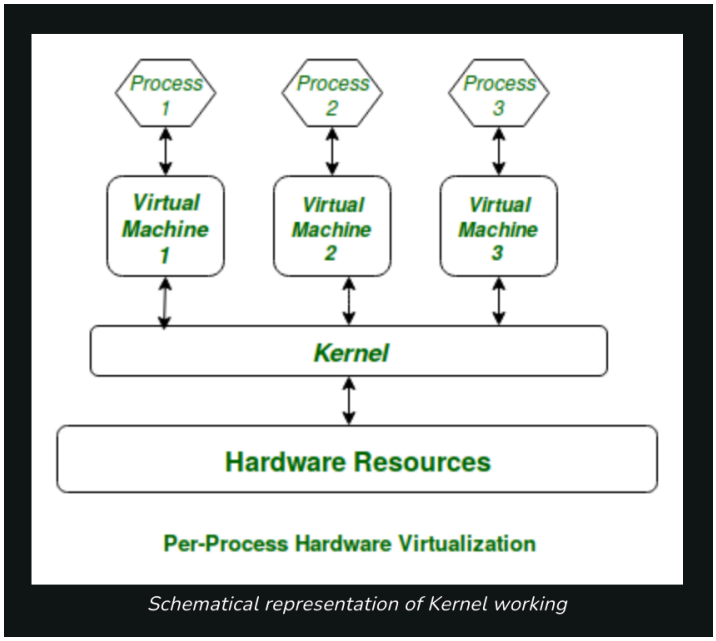
Introduction:



- The experiment aims to teach the creation, compilation, and management of Linux kernel modules through a Linux virtual machine.
- By writing and integrating kernel modules, we will gain hands-on experience in interacting directly with the kernel, understanding its benefits and risks.
- This project covers the process of writing a simple Linux kernel module, including the necessary header files and the use of the printk() function for logging information. It also explores the process of loading and unloading modules, as well as the importance of handling errors and memory management in kernel programming.
- The experiment emphasizes the significance of keeping code in user space whenever possible, and provides guidelines for when it may be necessary to write a kernel module.

Theory:





Schematical representation of Kernel working

The Linux Kernel: A Fundamental Component

- Understanding the role of the Linux kernel in managing system resources and facilitating communication between hardware and software layers.
- Examining the essential services provided by the Linux kernel, such as process management, memory allocation, and device drivers.

Extending the Linux Kernel with Modules

- Exploring kernel modules as dynamically loadable and unloadable code segments.
- Learning how modules extend the functionality of the Linux kernel without requiring a full system reboot.
- Comparing Linux's modular approach with monolithic kernels.

Key Aspects of Kernel Modules

- Examining module entry and exit points as designated functions that dictate the module's behavior during loading and unloading.
- Understanding the importance of the entry point function in initializing the module and preparing it for interaction with the kernel.
- Recognizing the role of the exit point function in facilitating cleanup operations during unloading.

Registering and Integrating Kernel Modules

- Learning how proper registration of module entry and exit points ensures seamless integration with the Linux kernel.
- Understanding the use of specialized macros to inform the kernel about the locations of the module's entry and exit functions.
- Ensuring system integrity and coherence by registering module entry and exit points.

Kernel Logging and System Management

- Exploring kernel logging mechanisms facilitated by the **printk()** function.
- Understanding the use of printk() as the kernel equivalent of printf().
- Learning how to manage log output for debugging, error tracking, and system monitoring.

- Utilizing command-line utilities like **dmesg** to gain insights into kernel activities and monitor system behavior in real-time.

Implementation:

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~$ cd coc
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ cat simple.c
#include <linux/init.h>
#include <linux/module.h>

static int __init simple_init(void) {
    printk(KERN_INFO "Simple module initialized\n");
    return 0;
}

static void __exit simple_exit(void) {
    printk(KERN_INFO "Simple module exited\n");
}

module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Abhi_Mehta 221080001");
```

simple.c > ...

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

static int __init simple_init(void) {
    printk(KERN_INFO "Simple module initialized\n");
    return 0;
}

static void __exit simple_exit(void) {
    printk(KERN_INFO "Simple module exited\n");
}

module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Abhi_Mehta 221080001");

```

Makefile:

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
```

Output:

Listing all the modules using lsmod command:

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ lsmod
Module                  Size  Used by
rfcomm                 98304  4
ccm                    20480  9
cmac                   12288  3
algif_hash             12288  1
algif_skcipher         12288  1
af_alg                 32768  6 algif_hash,algif_skcipher
bnep                   32768  2
intel_rapl_msr         20480  0
intel_rapl_common      40960  1 intel_rapl_msr
amdgpu              15589376  26
snd_sof_amd_rembrandt    16384  0
snd_sof_amd_renoir     16384  0
snd_sof_amd_acp        53248  2 snd_sof_amd_rembrandt,snd_sof_amd_renoir
snd_sof_pci            24576  2 snd_sof_amd_rembrandt,snd_sof_amd_renoir
snd_sof_xtensa_dsp     12288  1 snd_sof_amd_acp
snd_sof               360448  2 snd_sof_amd_acp,snd_sof_pci
snd_hda_codec_realtek   192512  1
snd_hda_codec_generic   122880  1 snd_hda_codec_realtek
snd_sof_utils          16384  1 snd_sof
snd_hda_codec_hdmi     94208  1
edac_mce_amd           40960  0
snd_soc_core          446464  1 snd_sof
snd_hda_intel          61440  4
snd_intel_dspcfg       32768  2 snd_hda_intel,snd_sof
snd_compress           28672  1 snd_soc_core
snd_intel_sdw_acpi     16384  1 snd_intel_dspcfg
ac97_bus               12288  1 snd_soc_core
kvm_amd               208896  0
snd_hda_codec         212992  4 snd_hda_codec_generic,snd_hda_codec_hdmi,snd_hda_intel,snd_hda_codec_realtek
snd_pcm_dmaengine      16384  1 snd_soc_core
iwlmvm                843776  0
snd_hda_core          147456  5 snd_hda_codec_generic,snd_hda_codec_hdmi,snd_hda_intel,snd_hda_codec,snd_hda_codec_realtek
snd_pci_ps             24576  0
kvm                  1409024  1 kvm_amd
snd_rpl_pci_acp6x      16384  0
snd_hwdep              20480  1 snd_hda_codec
binfmt_misc            24576  1
amdxcp                 12288  1 amdgpu
irqbypass              12288  1 kvm
snd_acp_pci            12288  0
iommu_v2               24576  1 amdgpu
snd_seq_midi           24576  0
```

Using make command to compile the module:

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ make
make -C /lib/modules/6.5.0-27-generic/build M=/home/abhi/coc modules
make[1]: Entering directory '/usr/src/linux-headers-6.5.0-27-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/abhi/coc/simple.o
  MODPOST /home/abhi/coc/Module.symvers
  CC [M]  /home/abhi/coc/simple.mod.o
  LD [M]  /home/abhi/coc/simple.ko
  BTF [M] /home/abhi/coc/simple.ko
Skipping BTF generation for /home/abhi/coc/simple.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.5.0-27-generic'
```

Loading and unloading the kernel module:

Loading the module:

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ sudo insmod simple.ko
[sudo] password for abhi:
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

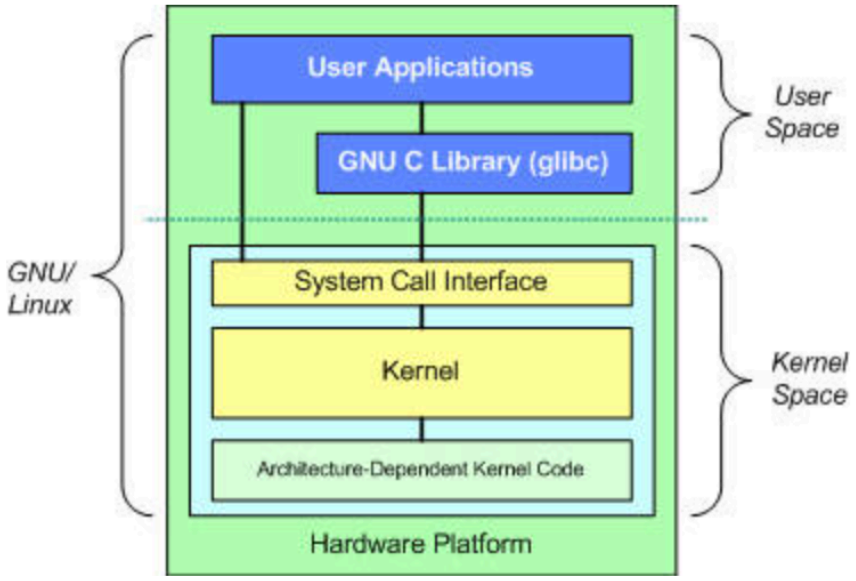
```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ lsmod | grep simple
simple                 12288  0
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ dmesg
dmesg: read kernel buffer failed: Operation not permitted
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ sudo dmesg -c
[    0.000000] Linux version 6.5.0-27-generic (buildd@lcy02-amd64-031) (x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0, GNU ld (GNU Bin
utils for Ubuntu) 2.38) #28~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Mar 15 10:51:06 UTC 2 (Ubuntu 6.5.0-27.28~22.04.1-generic 6.5.13)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-6.5.0-27-generic root=UUID=66d9261c-2ff3-4486-9566-7469f72f221c ro quiet splash vt.handoff=7
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] BIOS-provided physical RAM map:
```

```
[  663.924288] audit: type=1107 audit(1713766003.063:174): pid=823 uid=102 auid=4294967295 ses=4294967295 subj=unconfined msg='apparmor="DENIED" opera
tion="dbus_method_call"  bus="system" path="/org/freedesktop/RealtimeKit1" interface="org.freedesktop.RealtimeKit1" member="MakeThreadRealtime" mask="
send" name="org.freedesktop.RealtimeKit1" pid=7613 label="snap.telegram-desktop.telegram-desktop" peer_pid=1382 peer_label="unconfined"
               exe="/usr/bin/dbus-daemon" sauid=102 hostname=? addr=? terminal=?'
[  664.551080] audit: type=1326 audit(1713766003.687:175): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.telegram-desktop.telegram-desktop pid=7613 comm
="telegram-deskto" exe="/snap/telegram-desktop/5820/usr/bin/telegram-desktop" sig=0 arch=c000003e syscall=203 compat=0 ip=0x7e5be0990531 code=0x50000
[  664.551093] audit: type=1326 audit(1713766003.687:176): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.telegram-desktop.telegram-desktop pid=7613 comm
="telegram-deskto" exe="/snap/telegram-desktop/5820/usr/bin/telegram-desktop" sig=0 arch=c000003e syscall=203 compat=0 ip=0x7e5be0990531 code=0x50000
[  664.551100] audit: type=1326 audit(1713766003.687:177): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.telegram-desktop.telegram-desktop pid=7613 comm
="telegram-deskto" exe="/snap/telegram-desktop/5820/usr/bin/telegram-desktop" sig=0 arch=c000003e syscall=203 compat=0 ip=0x7e5be0990531 code=0x50000
[  664.551105] audit: type=1326 audit(1713766003.687:178): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.telegram-desktop.telegram-desktop pid=7613 comm
="telegram-deskto" exe="/snap/telegram-desktop/5820/usr/bin/telegram-desktop" sig=0 arch=c000003e syscall=203 compat=0 ip=0x7e5be0990531 code=0x50000
[ 1976.027821] simple: loading out-of-tree module taints kernel.
[ 1976.027829] simple: module verification failed: signature and/or required key missing - tainting kernel
[ 1976.028349] Simple module initialized
```

Unloading the module:

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ sudo rmmod simple
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ sudo dmesg -c
[ 2322.993438] Simple module exited
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

Key Points:



Kernel Modules

- Kernel modules are loadable kernel code that can be dynamically added or removed from the running kernel.
- They offer several advantages, including dynamic loading and unloading, efficient memory usage, and targeted optimizations.

The Role of Kernel Modules

- Kernel modules extend the functionality of the kernel without requiring a full system reboot.
- They enable developers to add or remove features as needed, making the system more flexible and adaptable to evolving needs and technological advancements.

Best Practices for Kernel Module Development

- Diligent error handling, resource management, and system stability maintenance are crucial to mitigate the risk of instability.
- Careful logging practices can help developers safeguard the stability of the system while harnessing the power of kernel modules.

Kernel Modules and the Open-Source Community

- Kernel modules are an essential part of the Linux operating system, enabling customization, modular design, and targeted optimizations.
- Understanding and working with kernel modules is valuable for kernel developers, device driver writers, system administrators, and anyone interested in exploring the inner workings of the Linux kernel.

Conclusion:

- Kernel modules offer a flexible and efficient mechanism for extending the functionality of the Linux kernel.
- By mastering module creation, loading, and unloading, developers can unlock new avenues for innovation while ensuring system stability and security.
- Embracing the dynamic nature of Linux development with a collaborative ecosystem of innovation and growth within the open-source community is key to harnessing the full potential of kernel modules.

—----------The End—----------