A PROJECT REPORT ON

Creating Linux Kernal Modules

Submitted by

GOLU KUMAR 231107140042

MD SARFUDDIN 231107140053

MD SAKIB 231107140055

MD HUSSIAN 231107140054

VIKKI RAJ 231107140034

in partial fulfillmentfor the award of the degree

of

DIPLOMA

in

Computer Science Engineering

DEPARTMENT OF C.S.E

SCHOOL OFVOCATIONAL EDUCATION

AND TRAINING

Paralakhemundi, 761211

CENTURION UNIVERSITY

OF

TECHNOLOGYAND MANAGEMENT

MARCH2025

SPECIMEN CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOLOF VOCATIONAL EDUCATION AND TRAINING

Paralakhemundi Campus

BONAFIDECERTIFICATE

Certified that this project report **CREATING LINUX KERNAL MODULES** is the Bonafide work of **"Md Hussain"** who carried out the project work under my supervision. This is to further certify to the best of my knowledge, that this project has not been carried outearlier in this institute and the university.

SIGNATURE

Harendra Kumar

Project Associate

Certified that the above-mentioned project has been duly carried out as per the norms of the college and statutes of the university.

SIGNATURE

HEAD OF THE DEPARTMENT / DEAN OF THESCHOOL

Professor of Computer Science and Engg.

DEPARTMENTSEAL

DECLARATION

I hereby declare that the project entitled **Creating Linux Kernal Modules** submitted for the "Project" of **4th** semester **DIPLOMA** in Computer Science and Engineering is my original work and the project has not formed the basis for the award of any Degree / Diploma or any other similar titles in any other University /Institute.

Name of the Student:

Signature of the Student:

Registration No:

Place:

Date:

ACKNOWLEDGEMENTS

I wish to express my profound and sincere gratitude to **Mr. Harendra Kumar,** SOVET CUTM, paralakhemundi Campus, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

I thank Mr. Laxmidhar Behra, Head of the Dept. of Computer Science and Engineering, SOVET, Paralakhemundi Campus and Dr. Prafulla Panda, Dean of School of Vocational Education and Training, Paralakhemundi Campus for extending their support during Course of this investigation.

Name of the Student:

Signature of the Student:

Registration No:

Place:

Date:

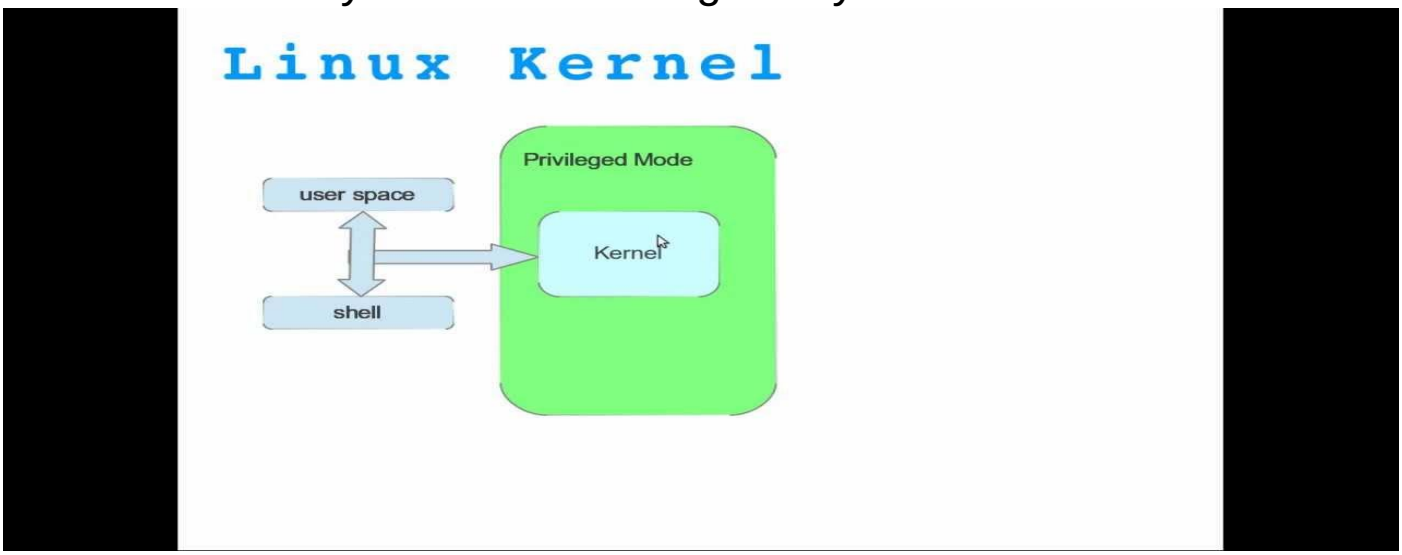
Contents

Introduction

Linux Kernel Modules (LKMs) are dynamically loadable pieces of code that extend the functionality of the Linux kernel without requiring a system reboot. They allow developers to add features like device drivers, file system extensions, and security modules without modifying the kernel source code. LKMs run in **kernel space**, giving them direct access to hardware and system resources, making them powerful yet risky if improperly handled.

To create a Linux kernel module, developers must write a C program including essential kernel headers like <linux/module.h> and <linux/init.h>. The module consists of an **initialization function**, which is executed when the module is loaded, and an **exit function**, which runs when it is removed. Compilation requires a **Makefile**, ensuring compatibility with the running kernel version. Modules can be loaded using insmod, removed using rmmod, and their logs monitored via dmesg.

Kernel modules are widely used in **device drivers, networking, security, and system monitoring**. However, incorrect implementation can crash the system, and debugging is more complex than user-space applications. Despite these challenges, LKMs offer a modular, flexible, and efficient approach to extending Linux functionality without rebootibg the system.

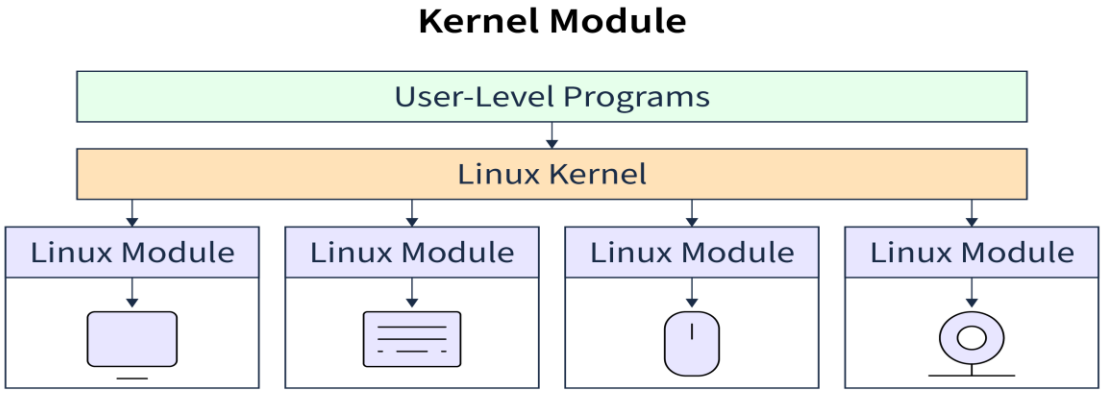


System Overview and Features

A Linux Kernel Module (LKM) operates in kernel space, allowing it to access low-level system resources. Unlike traditional user-space applications, kernel modules have direct access to hardware and system memory, making them highly efficient but also risky if not handled correctly. This project covers:

The basics of Linux kernel modules

- Writing and compiling a simple kernel module
- Loading and unloading the module
- Debugging kernel messages using dmesg
- Understanding kernel-space vs. user-space programming
- Handling kernel parameters and module interactions



SCALER Topics

Kernel Module Development

Kernel modules follow a specific structure and require kernel-specific functions for initialization and cleanup. Unlike user-space applications, they run with high privileges, making error handling critical.

1. Writing a Simple Kernel Module

Kernel modules must include specific headers and follow a structured format. Below is an example of a basic kernel module:

c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("A Simple Linux Kernel Module");
MODULE_VERSION("0.1");

 static int __init my_module_init(void) {
 printk(KERN_INFO "Kernel Module Loaded!\n");
 return 0;
 }

 static void __exit my_module_exit(void) {
 printk(KERN_INFO "Kernel Module Unloaded!\n");
 }
```

module_init(my_module_init);
module_exit(my_module_exit);

This module contains:
- module_init() to initialize the module upon loading
- module_exit() to handle cleanup upon removal
- printk() for kernel-level logging

2. Compiling the Kernel Module

Kernel modules require compilation against the currently running kernel version. Create a Makefile to automate the process:

```
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
```

This allows users to build the module using make and clean up compiled files using make clean.

3. Loading and Unloading the Module

To test the module, use the following commands:

- Compile: make
- Insert module: sudo insmod my_module.ko
- Check logs: dmesg | tail
- Remove module: sudo rmmod my_module

Verify removal: dmesg | tail

4. Handling Kernel Parameters

Kernel modules can accept parameters during loading. This enhances flexibility and allows configuration without recompilation. Example:

c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("A Parameterized Linux Kernel Module");

static int param_var = 0;
module_param(param_var, int, 0);
MODULE_PARM_DESC(param_var, "An integer parameter");

static int __init my_param_module_init(void) {
printk(KERN_INFO "Kernel Module Loaded with param_var = %d\n", param_var);
return 0;
}

static void __exit my_param_module_exit(void) {
 printk(KERN_INFO "Kernel Module Unloaded\n");
```

```
}
module_init(my_param_module_init);
module_exit(my_param_module_exit);
```

This module allows the user to specify param_var when loading the module using:

sh

```
sudo insmod my_module.ko param_var=5
```

5. Debugging and Kernel Logging

Debugging in kernel space is challenginsince traditionnal debugging tools like gdb are not directly available. Instead, we use:

- printk(KERN_DEBUG ...) for logging
- dmesg to view kernel logs
- /var/log/syslog for extended debugging

- **Advantages & Disadvantages**

Advantages

- Modular approach to kernel development
- No need to reboot the system for updates
- Helps in debugging and driver development
- Efficient and low-latency communication with hardware
- Extends kernel functionality dynamically

Disadvantages

- Requires understanding of kernel internals
- Incorrect code can crash the system
- Debugging is harder than user-space applications
- Security risks if improperly implemented

▪ Applications of Kernel Modules

Kernel modules are used in a variety of applications, including:

1. **Device Drivers**: Supporting hardware devices such as keyboards, printers, and network adapters.
2. **File System Extensions**: Adding new file systems or modifying existing ones.
3. **System Monitoring**: Collecting and logging system data for analysis.
4. **Security Modules**: Implementing custom access control and firewall features.
5. **Network Protocol Implementation**: Adding support for new networking protocols.

▪ Future Enhancements

- Implementing a character device driver as a module
- Creating an LKM that interacts with hardware (e.g., GPIO, USB)
- Exploring kernel synchronization techniques (mutex, spinlocks)
- Using dynamic kernel parameters for runtime configuration
- Developing a networking module for packet filtering
- Integrating with modern Linux kernel debugging tools

CONCLUSION

This project provides hands-on experience in writing, compiling, and loading Linux kernel modules. Through this process, developers gain insights into kernel-space programming, debugging techniques, and modular kernel development. Future work can focus on expanding this knowledge into device driver development and more advanced kernel extensions.

By understanding and implementing kernel modules, developers can create powerful and efficient system enhancements that interact closely with the Linux operating system. With further research, the knowledge gained from this project can be extended to complex kernel-based solutions such as real-time systems and security frameworks.

ASSESSMENT

Internal:

SL NO	RUBRICS	FULL MARK	MARKS OBTAINED	REMARKS
1	Understanding the relevance, scope and dimension of the project	10		
2	Methodology	10		
3	Quality of Analysis and Results	10		
4	Interpretations and Conclusions	10		
5	Report	10		
	Total	**50**		

Date: **Signature of the Faculty**

COURSE OUTCOME (COs) ATTAINMENT

➢ **Expected Course Outcomes (COs):**

(Refer to COs Statement in the Syllabus)

_____

_____

_____

_____

➢ **Course Outcome Attained:**

How would you rate your learning of the subject based on the specified COs?

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

1 2 3 4 5 6 7 8 9 10

LOW **HIGH**

➢ **Learning Gap (if any):**

_____

_____

_____

_____

➢ **Books / Manuals Referred:**

_____

_____

_____

_____

Date: **Signature of the Student**

➢ **Suggestions / Recommendations:**

(By the Course Faculty)

_____

_____

_____

_____

Date: **Signature of the Faculty**