WIPRO NGA Program – LDD Batch

Capstone Project Presentation – 31 July 2024

Project Title – Linux device driver for system metrics

Presented by – Vennam Srinivasareddy

Introduction

The Linux Device Driver Project aims to develop a character device driver that retrieves and displays system metrics, such as CPU usage, memory usage, and disk I/O, on a Linux system.

•**What we are doing:** Creating a character device driver, which is a program that allows the operating system to interact with hardware devices.

•**Why it's important:** Understanding how to write a device driver helps us learn about how the Linux kernel works and how it manages hardware.

•**What we will learn:** We'll gain hands-on experience with kernel programming, learn how device drivers are structured, and see how they handle input and output operations.

What is a device driver

A device driver is software that allows the operating system to communicate with hardware devices. It acts as an intermediary, translating system calls into hardware-specific commands.

Types of Device Drivers:

- **Character Drivers:** Handle data in a stream of characters (e.g., serial ports).
- **Block Drivers:** Manage data in fixed-size blocks (e.g., hard drives).
- **Network Drivers:** Interface with network hardware (e.g., Ethernet cards).
- **USB Drivers:** Control USB devices and manage connections.

Purpose of the Project

Purpose of the Project:

•**Create a functional character device driver:** Develop a basic driver that handles reading from and writing to a device.

•**Learn how to register and unregister device drivers:** Understand the process of making the driver known to the Linux kernel and removing it when no longer needed.

•**Understand basic file operations in device drivers:** Implement open, read, write, and release operations for the device.

•**Learn how user programs interact with kernel modules:** Understand the flow of data between user applications and the driver.

•**Improve debugging skills:** Gain experience in finding and fixing issues in kernel code.

•**Handle concurrent access to the device:** Learn how to manage multiple accesses to the device safely.

Motivation

The motivation for the Linux Device Driver Project is to give hands-on experience with Linux kernel programming and device driver creation. By working on this project, participants learn how to interact with the Linux kernel and retrieve system metrics like CPU and memory usage. This practical experience helps build a strong understanding of how software interacts with hardware. The project aims to enhance technical skills and prepare participants for advanced system development roles.

System Design

Module Operations: The driver can be loaded and unloaded, creating and removing the device file.

Device File: A character device file is used for communication between the driver and user space

File Operations: Implement functions to open, read, and close the device file.

Reading Metrics: Use kernel functions to get CPU, memory, and disk I/O metrics.

Timers: Set up timers to collect metrics periodically.

Concurrency and Cleanup: Handle multiple accesses safely and ensure all resources are freed on unload.

Error Handling: Ensure the driver handles errors gracefully and cleans up resources properly.

Module Implementation

Character Device Registration:
•**Module Used:** register_chrdev, unregister_chrdev
•Registers/unregisters a character device, assigning a major number for identification.
Device Creation:
•**Module Used:** class_create, device_create
•Creates a device class and a device within it for user space access.
Memory Allocation:
•**Module Used:** kmalloc, kfree
•Allocates and frees dynamic memory in the kernel.
Timer Setup:
•**Module Used:** timer_setup, mod_timer
•Sets up and modifies timers for periodic tasks.
System Information Retrieval:
•**Module Used:** si_meminfo, /proc filesystem
• Retrieves memory info and other system metrics from the kernel and /proc filesystem.

RPS learn..evolve

Testing and Validation

- **Test Cases:**
  - **Module Load/Unload:**
    - **Description:** Test loading and unloading the device driver module.
    - **Commands:** insmod sys_metrics.ko to load, rmmod sys_metrics.ko to unload.
    - **Expected Results:** Module should load and unload without errors. Check with dmesg for kernel log messages.
  - **Device File Creation:**
    - **Description:** Verify the creation of the device file.
    - **Commands:** ls /dev/sys_metrics
    - **Expected Results:** The device file /dev/sys_metrics should be present.
  - **Multiple Reads:**
    - **Description:** Test reading metrics data multiple times.
    - **Commands:** cat /dev/sys_metrics to read the data.
    - **Expected Results:** Each read should return the current system metrics without errors.
  - **Error Handling:**
    - **Description:** Test the device driver’s response to invalid operations.
    - **Commands:** echo "test" > /dev/sys_metrics (invalid write operation).
    - **Expected Results:** Invalid operation should be rejected, and appropriate error messages should appear in dmesg.

Performance Considerations

- **Efficient Timer Usage:**
  - •Use timers judiciously to avoid system overload.
- **Optimized Code:**
  - •Write concise and efficient code to enhance performance.
- **Minimal Memory Usage:**
  - •Allocate only necessary memory and free it promptly.
- **Low CPU Impact:**
  - •Ensure the driver uses minimal CPU resources.
- **Regular Profiling:**
  - •Continuously profile the driver to identify and resolve performance issues.

Challenges Faced

- **Kernel Space Complexity:**
- Understanding kernel space constraints.
- Solution: Studied kernel documentation, used printk for debugging.
- **Memory Management:**
- Efficient memory allocation and deallocation.
- Solution: Used kmalloc and kfree properly, ensured no memory leaks.
- **Synchronization Issues:**
- Handling concurrent access to data.
- Solution: Implemented spinlocks and mutexes.
- **Error Handling:**
- Managing errors without system crashes.
- Solution: Added error checking and dmesg logging.
- **Performance Optimization:**
- Maintaining system performance.
- Solution: Profiled and optimized code, minimized resource usage.

Conclusion

•**Summary of Achievements:**

•Successfully developed and tested a Linux device driver for system metrics.

•**Future Enhancements:**

•Expand metrics collection to include more system parameters and detailed data.

•**Performance Optimization:**

•Further refine the driver to minimize CPU and memory usage through profiling and optimizations.

•**User Interface Improvements:**

•Enhance user interaction by adding more features or a graphical interface.