WIPRO NGA Program – LDD Batch

Capstone Project Presentation – 31 July 2024

Project Title Here – Linux system metrics device driver

Presented by – Vishnu J P

INTRODUCTION

In modern computing environments, monitoring system performance and resource usage is critical for maintaining optimal operation and diagnosing issues. This project aims to provide a practical implementation of a Linux device driver that retrieves and reports system metrics. By developing a character device driver, we will gain hands-on experience with kernel programming, system monitoring, and character device interfaces.

MOTIVATION

The motivation behind developing a Linux System Metrics Device Driver is driven by the need for effective system monitoring, the desire for practical kernel programming experience, and the opportunity to address real-world challenges. By bridging the gap between user space and kernel space and addressing real-time performance needs, this project provides significant educational and professional value.

AIM OF THE PROJECT

The motivation behind developing a Linux System Metrics Device Driver is driven by the need for effective system monitoring, the desire for practical kernel programming experience, and the opportunity to address real-world challenges. By bridging the gap between user space and kernel space and addressing real-time performance needs, this project provides significant educational and professional value.

1. Develop a Functional Linux Device Driver

To design and implement a Linux device driver capable of retrieving and reporting system metrics such as CPU usage, memory usage, and disk I/O.

2. Facilitate System Performance Monitoring

To provide a tool for real-time monitoring of system performance metrics.
To enable users to access up-to-date information on system resource utilization.

3. Provide a Character Device Interface

To implement a character device that allows users to read system metrics from user space.

4. Demonstrate System Metrics Collection Techniques

To showcase methods for collecting and formatting system metrics using kernel APIs.

DEVICE DRIVER

In a Linux system, a device driver is a specialized software component that allows the operating system to interact with hardware devices. Drivers act as intermediaries between the hardware and the operating system, translating high-level commands from the OS into low-level operations that the hardware can understand.

Key Points About Device Drivers in Linux:

1. **Kernel Space and User Space:**

Device drivers operate in kernel space, meaning they run with high privileges and have direct access to the hardware.User-space applications interact with device drivers through system calls or APIs, but they do not have direct access to hardware.

2. **Types of Device Drivers:**

- **Character Drivers:** Handle data streams, such as those from keyboards, mice, or serial ports.
- **Block Drivers:** Manage data storage devices like hard drives and SSDs, dealing with data in blocks.
- **Network Drivers:** Manage network interfaces and protocols for communication over networks.

3. **Loading and Unloading:**

Drivers are often implemented as kernel modules, which can be loaded or unloaded from the kernel dynamically using commands like insmod, rmmod, and modprobe.

MODULES USED IN THIS PROJECT

Character Device Driver Module

This module serves as the core component of the project, enabling interaction with the Linux kernel through a character device. It handles the registration of the device, management of file operations, and communication between user space and kernel space.

Key Functions:

register_chrdev(): Registers the character device and assigns a major number.

unregister_chrdev(): Unregisters the character device.

file_operations structure: Defines the operations (e.g., read, write) that the driver supports.

Timer Module

The timer module is used for periodically updating system metrics. It allows the driver to schedule tasks to be executed at regular intervals.

Key Functions:

timer_setup(): Sets up a timer with a callback function.

mod_timer(): Modifies or adds a timer.

del_timer(): Deletes a timer.

Memory Management

Memory management functions are used to allocate and free memory for buffers and data structures in the kernel space.

Key Functions:

kmalloc(): Allocates memory in the kernel.

kfree(): Frees memory allocated by kmalloc().

File Operations

The file operations structure defines how the device interacts with file operations like read, write, and open.

Key Functions:

read(): Defines how data is read from the device.

write(): Defines how data is written to the device.

Error Handling

Implementing robust error handling mechanisms to manage and report errors effectively.

Key Functions:

printk(): Logs messages to the kernel log.

MODIFICATION AND ENHANCEMENT

User-Space Interface Improvements

Enhanced Usability: Improve the user-space interface for interacting with the device driver, making it more user-friendly and feature-rich.
Develop a graphical user interface (GUI) or command-line utilities for easier interaction.

Security Enhancements

Security Measures: Add security features to protect against unauthorized access and ensure the integrity of the metrics data.
Add data encryption or integrity checks to protect the metrics data.

Enhanced Performance Optimization

Optimized Performance: Optimize the performance of the driver to reduce its impact on system resources and improve efficiency.
Implement more efficient data collection and processing methods.

FUNCTIONS USING IN THIS PROJECT

1. **register_chrdev()**

Registers a character device with the kernel, allowing the device to be accessed from user space.
int register_chrdev(unsigned int major, const char *name, const struct file_operations *fops);

2. **unregister_chrdev()**

Unregisters a character device, freeing the major number and cleaning up associated resources.
void unregister_chrdev(unsigned int major, const char *name);

3. **class_create()**

Creates a device class, which is used to manage devices under /sys/class/.
struct class *class_create(struct module *owner, const char *name);

4. **device_create()**

Creates a device and registers it with sysfs, making it available as a device file under /dev/.
struct device *device_create(struct class *cls, struct device *parent, dev_t devt, void *drvdata, const char *fmt, ...);

5. **device_destroy()**

Destroys a device created with device_create(), cleaning up the device file.
void device_destroy(struct class *cls, dev_t devt);

learn..evolve

DRIVER COMPILATION

Steps to Compile:

- Create a Makefile.
- Run make in the terminal to build the kernel module (driver_name.ko)

Loading the Driver

Insert the Module:

- Command: sudo insmod driver_name.ko
- This command loads the driver into the kernel

Verify Module:

- Check loaded modules with lsmod | grep driver_name
- Inspect kernel logs with dmesg | grep driver_name

Unloading the Driver

Remove the Module:

- Command: sudo rmmod driver_name
- This command removes the driver from the kernel

Verify Removal:

- Check if the module is unloaded with lsmod | grep driver_name
- Inspect kernel logs with dmesg | grep driver_name

System Metrics Retrieval Methods

Memory Metrics:

- Function: si_meminfo()
- It retrieves current memory information from the system
- Data Format:Total RAM, Free RAM, Cached RAM

CPU Usage:

- Method: Parse /proc/stat
- Description: Extracts CPU usage statistics
- Data Format:CPU times (user, system, idle)

learn..evolve

THANK YOU