PROJECT NAME:

Linux Character Device Driver for System Metrics

INTRODUCTION

This project involves developing a Linux character device driver to retrieve and output system metrics such as CPU usage, memory usage, and disk I/O metrics. The device driver can be loaded and unloaded from the kernel and provides a character device interface for user interaction.

Source Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/slab.h>
#include <linux/proc_fs.h>
#include <linux/timer.h>
#include <linux/sched.h>
#include <linux/jiffies.h>

#define DEVICE_NAME "sys_metrics" // Device name
#define CLASS_NAME "sys_class"    // Device class name

static int majorNumber;                  // Major number for the device
```

```
static struct class* sysMetricsClass = NULL;   // Device class pointer
static struct device* sysMetricsDevice = NULL; // Device pointer

static struct timer_list metrics_timer; // Timer for periodic metrics collection
static unsigned long interval = 5 * HZ; // Interval for the timer (5 seconds)

static char *metrics_buffer; // Buffer to store the metrics
static int buffer_size = 0;  // Size of the buffer

// Function prototypes for device operations
static int dev_open(struct inode *inodep, struct file *filep);
static int dev_release(struct inode *inodep, struct file *filep);
static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t *offset);
static ssize_t dev_write(struct file *filep, const char *buffer, size_t len, loff_t *offset);

// File operations structure
static struct file_operations fops = {
    .open = dev_open,
    .read = dev_read,
    .write = dev_write,
    .release = dev_release,
};
// Device open function
static int dev_open(struct inode *inodep, struct file *filep) {
    printk(KERN_INFO "sys_metrics: Device opened\n");
    return 0;
}
// Device release function
```

```
static int dev_release(struct inode *inodep, struct file *filep) {
    printk(KERN_INFO "sys_metrics: Device closed\n");
    return 0;
}
// Device read function
static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t *offset) {
    int error_count = 0;

    // Check if the buffer is large enough
    if (len < buffer_size) return -EFAULT;

    // Copy the metrics data to user space
    error_count = copy_to_user(buffer, metrics_buffer, buffer_size);

    if (error_count == 0) {
        printk(KERN_INFO "sys_metrics: Sent %d characters to the user\n", buffer_size);
        return buffer_size;
    } else {
        printk(KERN_INFO "sys_metrics: Failed to send %d characters to the user\n",
error_count);
        return -EFAULT;
    }
}
// Device write function (not supported)
static ssize_t dev_write(struct file *filep, const char *buffer, size_t len, loff_t *offset) {
    printk(KERN_ALERT "sys_metrics: Write operation not supported\n");
    return -EINVAL;
}
// Function to collect system metrics
```

```
void get_system_metrics(void) {
    struct sysinfo sys_info;
    int num_cpus = num_online_cpus(); // Number of CPU cores
    int i;

    // Get memory information
    si_meminfo(&sys_info);

    // Format the metrics data into the buffer
    buffer_size = sprintf(metrics_buffer,
                          "CPU cores: %d\n"
                          "Total RAM: %lu kB\n"
                          "Free RAM: %lu kB\n"
                          "Shared RAM: %lu kB\n"
                          "Buffered RAM: %lu kB\n"
                          "Total Swap: %lu kB\n"
                          "Free Swap: %lu kB\n"
                          "Uptime: %lu seconds\n",
                          num_cpus,
                          sys_info.totalram * 4,
                          sys_info.freeram * 4,
                          sys_info.sharedram * 4,
                          sys_info.bufferram * 4,
                          sys_info.totalswap * 4,
                          sys_info.freeswap * 4,
                          sys_info.uptime);

    // Append CPU idle time for each core (this part needs proper implementation)
```

```
    for (i = 0; i < num_cpus; i++) {
        buffer_size += sprintf(metrics_buffer + buffer_size,
                               "CPU %d: %lu\n",
                               i,
                               get_cpu_idle_time_us(i));
    }
}
// Timer callback function
static void metrics_timer_callback(struct timer_list *timer) {
    printk(KERN_INFO "sys_metrics: Timer callback executed\n");
    get_system_metrics();
    mod_timer(&metrics_timer, jiffies + interval); // Restart the timer
}
// Module initialization function
static int __init sys_metrics_init(void) {
    printk(KERN_INFO "sys_metrics: Initializing the sys_metrics LKM\n");

    // Register the character device driver
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    if (majorNumber < 0) {
        printk(KERN_ALERT "sys_metrics failed to register a major number\n");
        return majorNumber;
    }
    printk(KERN_INFO "sys_metrics: registered correctly with major number %d\n",
majorNumber);

    // Register the device class
    sysMetricsClass = class_create(THIS_MODULE, CLASS_NAME);
    if (IS_ERR(sysMetricsClass)) {
```

```
        unregister_chrdev(majorNumber, DEVICE_NAME);
        printk(KERN_ALERT "Failed to register device class\n");
        return PTR_ERR(sysMetricsClass);
    }
    printk(KERN_INFO "sys_metrics: device class registered correctly\n");

    // Create the device
    sysMetricsDevice = device_create(sysMetricsClass, NULL, MKDEV(majorNumber, 0),
NULL, DEVICE_NAME);
    if (IS_ERR(sysMetricsDevice)) {
        class_destroy(sysMetricsClass);
        unregister_chrdev(majorNumber, DEVICE_NAME);
        printk(KERN_ALERT "Failed to create the device\n");
        return PTR_ERR(sysMetricsDevice);
    }
    printk(KERN_INFO "sys_metrics: device created correctly\n");

    // Allocate memory for the metrics buffer
    metrics_buffer = kmalloc(1024, GFP_KERNEL);
    if (!metrics_buffer) {
        device_destroy(sysMetricsClass, MKDEV(majorNumber, 0));
        class_destroy(sysMetricsClass);
        unregister_chrdev(majorNumber, DEVICE_NAME);
        printk(KERN_ALERT "sys_metrics: Failed to allocate memory\n");
        return -ENOMEM;
    }

    // Setup and start the timer
    timer_setup(&metrics_timer, metrics_timer_callback, 0);
```

```
    mod_timer(&metrics_timer, jiffies + interval);

    printk(KERN_INFO "sys_metrics: Device driver loaded successfully\n");
    return 0;
}
// Module exit function
static void __exit sys_metrics_exit(void) {
    del_timer(&metrics_timer); // Delete the timer
    kfree(metrics_buffer); // Free the buffer memory
    device_destroy(sysMetricsClass, MKDEV(majorNumber, 0)); // Remove the device
    class_unregister(sysMetricsClass); // Unregister the device class
    class_destroy(sysMetricsClass); // Destroy the device class
    unregister_chrdev(majorNumber, DEVICE_NAME); // Unregister the character device driver
    printk(KERN_INFO "sys_metrics: Device driver unloaded successfully\n");
}

// Specify the module initialization and exit functions
module_init(sys_metrics_init);
module_exit(sys_metrics_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("SOURAV");
MODULE_DESCRIPTION("A Linux character device driver to output system metrics.");
MODULE_VERSION("1.0");
```

Makefile:

```
obj-m += sys_metrics.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
```

Functional Requirements:

Device Driver:

- Load and unload from the kernel.
- Use kernel APIs to retrieve metrics.
- IOCTL for CPU core averages, memory, and disk usage.

System Metrics:

- Retrieve CPU, memory, and disk I/O metrics.
- Accurate and efficient data collection.

System Metrics Details

Data Sources:

- /proc filesystem for metrics.
- Define data format and structure for system info.

Metrics:

- CPU usage (by core).
- Memory usage.
- Disk I/O.

Character Device Interface:

Implementation:

- Readable character device for outputting metrics.

- User space interface for reading metrics data.

User Interface:

Commands:

- Load device driver.
- Unload device driver.
- Read metrics by reading from the character device.

DOCUMENTATION:

Code Comments and Explanations:

- The source code contains inline comments explaining the purpose of each function, variable, and key section of code.
- Key components include the get_system_metrics function for collecting system metrics, the metrics_timer_callback for handling periodic metric collection, and standard file operations (dev_open, dev_read, dev_write, dev_release).

User Manual:

Compiling the Device Driver:

1. Ensure you have the necessary kernel headers installed:

   sudo apt-get install linux-headers-$(uname -r)

2. Use the provided makefile to compile the driver:

   make

Loading the Device Driver:

1. Load the module into the kernel:

sudo insmod sys_metrics.ko

2. Verify the device file has been created:

   ls /dev/sys_metrics

Reading System Metrics:

1. Use the cat command to read from the device file:

   cat /dev/sys_metrics

Unloading the Device Driver:

1. Remove the module from the kernel:

   sudo rmmod sys_metrics

2. Verify the device file has been deleted:

   ls /dev/sys_metrics

Description of System Metrics Retrieval Methods:

System metrics are retrieved using various kernel APIs:

- si_meminfo(&sys_info) is used to gather memory information.
- get_cpu_idle_time_us(cpu_id) (or similar function) is used to gather CPU usage information.

The metrics_timer_callback function periodically collects and updates these metrics.