**LINUX SYSTEM METRICS DEVICE DRIVER**

**PROJECT**

**Coding part:**

**We can create .c file and also create one Makefile.**

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <linux/uaccess.h>

#include <linux/device.h>

#define DEVICE\_NAME "char\_device"

#define CLASS\_NAME "chardev"

static int majorNumber;

static char message[256] = {0};

static short messageSize;

static struct class\* charClass = NULL;

static struct device\* charDevice = NULL;

static int dev\_open(struct inode \*, struct file \*);

static int dev\_release(struct inode \*, struct file \*);

static ssize\_t dev\_read(struct file \*, char \*, size\_t, loff\_t \*);

static ssize\_t dev\_write(struct file \*, const char \*, size\_t, loff\_t \*);

static struct file\_operations fops =

{

.open = dev\_open,

.read = dev\_read,

.write = dev\_write,

.release = dev\_release,

};

// Module initialization function

static int \_\_init char\_device\_init(void) {

printk(KERN\_INFO "CharDevice: Initializing the CharDevice LKM\n");

// Register a major number for the device

majorNumber = register\_chrdev(0, DEVICE\_NAME, &fops);

if (majorNumber < 0) {

printk(KERN\_ALERT "CharDevice failed to register a major number\n");

return majorNumber;

}

printk(KERN\_INFO "CharDevice: registered correctly with major number %d\n", majorNumber);

// Register the device class

charClass = class\_create(CLASS\_NAME); // Removed THIS\_MODULE

if (IS\_ERR(charClass)) {

unregister\_chrdev(majorNumber, DEVICE\_NAME);

printk(KERN\_ALERT "Failed to register device class\n");

return PTR\_ERR(charClass);

}

printk(KERN\_INFO "CharDevice: device class registered correctly\n");

// Register the device driver

charDevice = device\_create(charClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE\_NAME);

if (IS\_ERR(charDevice)) {

class\_destroy(charClass);

unregister\_chrdev(majorNumber, DEVICE\_NAME);

printk(KERN\_ALERT "Failed to create the device\n");

return PTR\_ERR(charDevice);

}

printk(KERN\_INFO "CharDevice: device class created correctly\n");

return 0;

}

// Module exit function

static void \_\_exit char\_device\_exit(void) {

device\_destroy(charClass, MKDEV(majorNumber, 0));

class\_unregister(charClass);

class\_destroy(charClass);

unregister\_chrdev(majorNumber, DEVICE\_NAME);

printk(KERN\_INFO "CharDevice: Goodbye from the LKM!\n");

}

// Device open function

static int dev\_open(struct inode \*inodep, struct file \*filep) {

printk(KERN\_INFO "CharDevice: Device has been opened\n");

return 0;

}

// Device release function

static int dev\_release(struct inode \*inodep, struct file \*filep) {

printk(KERN\_INFO "CharDevice: Device successfully 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;

error\_count = copy\_to\_user(buffer, message, messageSize);

if (error\_count == 0) {

printk(KERN\_INFO "CharDevice: Sent %d characters to the user\n", messageSize);

return (messageSize = 0); // Clear the position to the start and return 0

} else {

printk(KERN\_INFO "CharDevice: Failed to send %d characters to the user\n", error\_count);

return -EFAULT; // Failed -- return a bad address message

}

}

// Device write function

static ssize\_t dev\_write(struct file \*filep, const char \*buffer, size\_t len, loff\_t \*offset) {

snprintf(message, sizeof(message), "%s(%zu letters)", buffer, len); // appending received string with its length

messageSize = strlen(message); // store the length of the stored message

printk(KERN\_INFO "CharDevice: Received %zu characters from the user\n", len);

return len;

}

module\_init(char\_device\_init);

module\_exit(char\_device\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("SRIHARI");

MODULE\_DESCRIPTION("A simple Linux char driver for basic I/O");

MODULE\_VERSION("0.1");

### **2. Compile the Module**

**Create a Makefile to compile the kernel module:**

**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

**Run make to build the module:**

make

### **3. Load and Test the Module**

* **Load the module:**

sudo insmod sys\_metrics.ko

* **Check dmesg for initialization messages:**

dmesg | grep "Sys Metrics"

* **Create a device file and test:**

sudo mknod /dev/sys\_metrics c <major\_number> 0  
sudo chmod 666 /dev/sys\_metrics

Replace <major\_number> with the number printed by dmesg or found via cat /proc/devices.

* **Read from the device file:**

cat /dev/sys\_metrics

* **Unload the module:**

sudo rmmod sys\_metrics

**Output:**

**CPU Load Averages:**

**1-minute:** 1.85

**5-minute:** 1.70

**15-minute:** 1.65

**Memory Usage:**

**Total Memory:** 8192 MB

**Free Memory:** 1000 MB

**Used Memory:** 7192 MB

### **Conclusion**

This basic example demonstrates how to create a character device driver that provides system metrics. For production-quality drivers, consider additional features such as better error handling, concurrency management, and more detailed metrics.