FINAL CAPSTONE PROJECT: Linux device driver for System metrics

Source code

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <linux/uaccess.h>

#include <linux/device.h>

#include <linux/timer.h>

#include <linux/slab.h>

#define DEVICE\_NAME "char\_device"

#define CLASS\_NAME "chardev"

static int majorNumber;

static char \*message;

static short messageSize;

static struct class \*charClass = NULL;

static struct device \*charDevice = NULL;

static struct timer\_list metrics\_timer;

static int cpu\_usage;

static int mem\_usage;

static int disk\_io;

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

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

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

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

static void update\_metrics(struct timer\_list \*t);

static struct file\_operations fops = {

.open = metrics\_dev\_open,

.read = metrics\_dev\_read,

.write = metrics\_dev\_write,

.release = metrics\_dev\_release,

};

static int \_\_init metrics\_init(void) {

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

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

if (majorNumber < 0) {

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

return majorNumber;

}

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

charClass = class\_create(CLASS\_NAME);

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 "MetricsDevice: device class registered correctly\n");

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 "MetricsDevice: device class created correctly\n");

timer\_setup(&metrics\_timer, update\_metrics, 0);

mod\_timer(&metrics\_timer, jiffies + msecs\_to\_jiffies(1000)); // 1 second interval

message = kmalloc(256, GFP\_KERNEL);

if (!message) {

printk(KERN\_ALERT "Failed to allocate memory for message buffer\n");

return -ENOMEM;

}

return 0;

}

static void \_\_exit metrics\_exit(void) {

del\_timer(&metrics\_timer);

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

class\_unregister(charClass);

class\_destroy(charClass);

unregister\_chrdev(majorNumber, DEVICE\_NAME);

kfree(message);

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

}

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

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

return 0;

}

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

printk(KERN\_INFO "MetricsDevice: Device successfully closed\n");

return 0;

}

static ssize\_t metrics\_dev\_read(struct file \*filep, char \*buffer, size\_t len, loff\_t \*offset) {

int error\_count = 0;

if (\*offset > 0) {

return 0;

}

snprintf(message, 256, "CPU Usage: %d%%, Memory Usage: %d%%, Disk I/O: %d\n", cpu\_usage, mem\_usage, disk\_io);

messageSize = strlen(message);

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

if (error\_count == 0) {

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

\*offset += messageSize;

return messageSize;

} else {

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

return -EFAULT;

}

}

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

snprintf(message, 256, "%s(%zu letters)", buffer, len);

messageSize = strlen(message);

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

return len;

}

static void update\_metrics(struct timer\_list \*t) {

cpu\_usage = 20; // Placeholder value

mem\_usage = 30; // Placeholder value

disk\_io = 40; // Placeholder value

mod\_timer(&metrics\_timer, jiffies + msecs\_to\_jiffies(1000));

}

module\_init(metrics\_init);

module\_exit(metrics\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Name");

MODULE\_DESCRIPTION("A simple Linux char driver for system metrics");

MODULE\_VERSION("0.1");