#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/seq\_file.h>

#include <linux/jiffies.h>

#include <linux/sched/signal.h>

#include <linux/device.h>

#include <linux/mm.h> // For si\_meminfo

#include <linux/timer.h>

#define DEVICE\_NAME "sysmetrics"

#define CLASS\_NAME "sysmetrics"

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Your Name");

MODULE\_DESCRIPTION("A Linux device driver to report system metrics");

MODULE\_VERSION("0.1");

static int major\_number;

static struct class\* sysmetrics\_class = NULL;

static struct device\* sysmetrics\_device = NULL;

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

return 0;

}

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

return 0;

}

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

int err\_count = 0;

char \*msg;

int msg\_len;

struct sysinfo info;

si\_meminfo(&info);

msg\_len = snprintf(NULL, 0, "Uptime: %llu seconds\nTotal RAM: %lu MB\nFree RAM: %lu MB\n",

(unsigned long long)(jiffies\_to\_msecs(get\_jiffies\_64()) / 1000),

info.totalram >> 20,

info.freeram >> 20) + 1;

msg = kmalloc(msg\_len, GFP\_KERNEL);

if (!msg) {

return -ENOMEM;

}

snprintf(msg, msg\_len, "Uptime: %llu seconds\nTotal RAM: %lu MB\nFree RAM: %lu MB\n",

(unsigned long long)(jiffies\_to\_msecs(get\_jiffies\_64()) / 1000),

info.totalram >> 20,

info.freeram >> 20);

err\_count = copy\_to\_user(buffer, msg, msg\_len);

kfree(msg);

if (err\_count == 0) {

return msg\_len;

} else {

return -EFAULT;

}

}

static struct file\_operations fops = {

.open = dev\_open,

.read = dev\_read,

.release = dev\_release,

};

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

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

if (major\_number < 0) {

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

return major\_number;

}

sysmetrics\_class = class\_create(CLASS\_NAME);

if (IS\_ERR(sysmetrics\_class)) {

unregister\_chrdev(major\_number, DEVICE\_NAME);

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

return PTR\_ERR(sysmetrics\_class);

}

sysmetrics\_device = device\_create(sysmetrics\_class, NULL, MKDEV(major\_number, 0), NULL, DEVICE\_NAME);

if (IS\_ERR(sysmetrics\_device)) {

class\_destroy(sysmetrics\_class);

unregister\_chrdev(major\_number, DEVICE\_NAME);

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

return PTR\_ERR(sysmetrics\_device);

}

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

return 0;

}

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

device\_destroy(sysmetrics\_class, MKDEV(major\_number, 0));

class\_unregister(sysmetrics\_class);

class\_destroy(sysmetrics\_class);

unregister\_chrdev(major\_number, DEVICE\_NAME);

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

}

module\_init(sysmetrics\_init);

module\_exit(sysmetrics\_exit);