ECE 373 - Embedded Operating Systems & Device Drivers **Spring 24**
Assignment #2 Report Wail (Wa'el) AL KALBANI

Objective:
Develop a Linux kernel module and a userspace program for managing a character device.

Project Deliverables (Github):

1. Kernel Module: **char_dev.c**
2. Userspace Program: **userspace.c**
3. Build System: **Makefile** (compiling both the module and test program)
4. Compilation Script: **typescript**

Part 1: Setting up the char device

(a) Using umount Command

- After loading your module, determine the major number assigned by checking /proc/devices.
- Create a device file using mknod:
  sudo mknod /dev/char1 c 237 0
- Permissions setup :
  sudo chmod 666 /dev/char1

(b) Using class_create and device_create

These functions are used within the kernel module to automate the creation of device files and to integrate with the sysfs, enhancing the module's interaction with the userspace:

- **Include device headers:**

```
#include <linux/device.h> // Provides definitions for device classes and functions
```

- **Global variables for class and device:**

```
static struct class *char1_class = NULL;
static struct device *char1_device = NULL;
```

- **Modifying initialization function:**

```
static int __init char1_init(void)
{
    printk(KERN_INFO "char1 module loading... initial_val=%d\n", initial_val);

    if (alloc_chrdev_region(&mydev_node, 0, DEVCNT, DEVNAME)) {
        printk(KERN_ERR "alloc_chrdev_region() failed!\n");
        return -1;
    }

    /* Create a class */
    char1_class = class_create(THIS_MODULE, "charclass");
    if (IS_ERR(char1_class)) {
        unregister_chrdev_region(mydev_node, DEVCNT);
        return PTR_ERR(char1_class);
    }

    /* Initialize the character device and add it to the kernel */
    cdev_init(&mydev.my_cdev, &mydev_fops);
    mydev.my_cdev.owner = THIS_MODULE;


    if (cdev_add(&mydev.my_cdev, mydev_node, DEVCNT)) {
        printk(KERN_ERR "cdev_add() failed!\n");
        class_destroy(char1_class);
        unregister_chrdev_region(mydev_node, DEVCNT);
        return -1;
    }

    /* Create the device */
    char1_device = device_create(char1_class, NULL, mydev_node,
NULL, DEVNAME);
    if (IS_ERR(char1_device)) {
        cdev_del(&mydev.my_cdev);
        class_destroy(char1_class);
        unregister_chrdev_region(mydev_node, DEVCNT);
        return PTR_ERR(char1_device);
    }

    return 0;
}
```

- **Modifying cleanup function**

```
static void __exit char1_exit(void)
{
    device_destroy(char1_class, mydev_node);
    cdev_del(&mydev.my_cdev);
    class_destroy(char1_class);
    unregister_chrdev_region(mydev_node, DEVCNT);
    printk(KERN_INFO "char1 module unloaded!\n");
}
```

Part 2: Hooking up file operations

- Added a `printk()` statement in `char1_read` to log when the function is entered:

```
printk(KERN_INFO "In the raed() system call");
```

- Added a `printk()` statement in `char1_write` to log when the function is entered:

```
printk(KERN_INFO "In the write() system call");
```

These modifications clearly log entry into the respective system calls, providing immediate feedback that these functions are being executed.

Part 3: Userspace testing program - userspace.c

- **Opening the Character Device**
  - Open the character device /dev/char1 to begin interaction.

```
// Open the device file
 fd_device = open(PATH_DEVICE, O_RDWR);
 if (fd_device < 0) {
     perror("Failed to open the device");
     return errno;
 }
```

- **Reading the Initial Value**
  - Read the initial value of syscall_val from the device to verify the current state before modification.

```
// Read the initial value from the device
 if (read(fd_device, &input_val, sizeof(input_val)) < 0) {
     perror("Failed to read from device");
     close(fd_device);
     return errno;
 }
 printf("Initial value read from device: %d\n", input_val);
```

- **Writing a New Value**
  - Write a new value to syscall_val to test the write capability of the device.
  - Prompt the user for a new value.
  - Write this value to the device.

```
// Write the new value to the device
  if (write(fd_device, &new_val, sizeof(new_val)) < 0) {
      perror("Failed to write to device");
      close(fd_device);
      return errno;
  }
  printf("New value written to device: %d\n", new_val);
```

- **Reading Back the Value After Write**
  - Read back the value of syscall_val after the write operation to confirm that the write was successful.

```
// Read back the value from the device to confirm the write
 if (read(fd_device, &input_val, sizeof(input_val)) < 0) {
     perror("Failed to read from device after write");
     close(fd_device);
     return errno;
 }
 printf("Value read from device after write: %d\n", input_val);
```

- **Resource Cleanup**
  - All resources are properly released after operations, especially upon unloading.

```
// Clean up and close the device file
 close(fd_device);
```

Part 4: Changes

- **Modifying kernel module (char_dev.c) to accept a parameter**
  - Allow the initial value of syscall_val to be set through a module parameter.

```
/* this shows up under /sys/modules/char1/parameters */
static int initial_val = 42;    //User parameter input
module_param(initial_val, int, S_IRUSR | S_IWUSR);
MODULE_PARM_DESC(initial_val, "Initial value for the character device");
```

  - module_param macro is used to declare initial_val as a module parameter that can be set at load time.
  - S_IRUGO | S_IWUSR sets the parameter to be both readable and writable from user space.
- **Reloading the module with a new initial value**
  - Load the kernel module with a user-defined initial value from the command line

Bash command:

```
sudo rmmod char1  # Unload the module if it's already loaded
sudo insmod char1.ko initial_val=97  # Load the module with a new val
```

- **Testing the new initial value with the userspace program**
  - userspace program is executed using the same Makefile.
  - When running ./userspace, /dev/char1 opens , reads, and displays the initial value.
  - The program also writes a new value and then reads it back to verify.
- Reading the module parameter from sysfs
  - This is to confirm the current value of the module parameter from sysfs
  - In terminal:

```
cat /sys/module/char1/parameters/initial_val
```