CS3003D : OPERATING SYSTEMS (S5 Btech)

Name: Shifana S Shafeek
Roll Number: B221204CS

ASSIGNMENT 2

Assignment : Character Device Driver

➔ Problem Statement

Objective:
Create a Character Device Driver with the following functionality:

1. ***Kernel Version Check***:

• The driver must accept an array parameter called kernel_version, which specifies the current kernel version.

• The driver should only be inserted if the provided kernel version matches the version used to compile the module.

• For example, if the module was compiled for version 6.5.1, it should only be successfully inserted if you run
insmod <your_driver_name>.ko kernel_version=6,5,1

Character_Device_Driver.c

Character_Device_Driver.c

Character_Device_Driver.c > ...

```c
#include<linux/init.h>
#include<linux/module.h>
#include<linux/moduleparam.h>
#include<linux/fs.h>
#include <linux/uaccess.h>
#include <linux/string.h>
#include <linux/utsname.h>


MODULE_LICENSE("GPL");

#define BUFFER_SIZE 1024


static int kernel_version[3];

static int open_count = 0;
static char kernel_buffer[BUFFER_SIZE];
int kernel_version_len = 3;
module_param_array(kernel_version, int, &kernel_version_len, 0644);
int major_num = 0;


// Function prototypes for file operations
ssize_t dev_read(struct file *pfile, char __user *user_buffer, size_t len, loff_t *offset);
ssize_t dev_write(struct file *pfile, const char __user *user_buffer, size_t len, loff_t *offset);
int dev_open(struct inode *pinode, struct file *pfile);
int dev_close(struct inode *pinode, struct file *pfile);

/*To hold the file operation on this device*/
    ssize_t dev_read(struct file *pfile, char __user *user_buffer, size_t len, loff_t *offset){
        printk(KERN_INFO "Read function called!\n");
        size_t bytes_to_read = strlen(kernel_buffer);


            if (*offset >= bytes_to_read) {
                return 0;
            }

        if (len > bytes_to_read - *offset) {
            len = bytes_to_read - *offset;
        }

        if (copy_to_user(user_buffer, kernel_buffer + *offset, len)) {
                return -EFAULT;
        }

        *offset += len;
        printk(KERN_INFO "Sent to user: %s\n", kernel_buffer);
        return len;
    }

        ssize_t dev_write(struct file *pfile, const char __user *user_buffer, size_t len,loff_t *offset){
            printk(KERN_INFO "Write function called!\n");
             if (len > BUFFER_SIZE - 1) {
                return -EFAULT;
            }
            if (copy_from_user(kernel_buffer, user_buffer, len)) {
                return -EFAULT;
            }
            kernel_buffer[len] = '\0';
            printk(KERN_INFO "Received from user: %s\n", kernel_buffer);
            return len;

        }

    int dev_open(struct inode *pinode, struct file *pfile){
        open_count++;
            printk(KERN_INFO "Device opened %d'th times\n", open_count);
            return 0;
    }

    int dev_close(struct inode *pinode, struct file *pfile){
        printk(KERN_INFO "Device closed\n");
            return 0;
    }
```

```c

struct file_operations char_driver_file_operations = {
    .owner   = THIS_MODULE,
    .open    = dev_open,
    .read    = dev_read,
    .write   = dev_write,
    .release = dev_close,
};


static int __init Character_Device_Driver_init(void){
    int running_version[3];
    struct new_utsname *uts;

    uts = utsname();

    sscanf(uts->release, "%d.%d.%d", &running_version[0], &running_version[1], &running_version[2]);
    printk(KERN_ALERT "kernal : %d,%d,%d",kernel_version[0], kernel_version[1], kernel_version[2]);
    if (kernel_version[0] != running_version[0] ||
        kernel_version[1] != running_version[1] ||
        kernel_version[2] != running_version[2]) {
        printk(KERN_ALERT "Kernel Version Mismatch: Expected %d.%d.%d, but got %d.%d.%d\n",
               running_version[0], running_version[1], running_version[2],
               kernel_version[0], kernel_version[1], kernel_version[2] );
        return -EINVAL;
    }

    major_num = register_chrdev(0, "Character_Driver", &char_driver_file_operations);
    if (major_num < 0) {
        printk(KERN_ALERT "Failed to register character driver\n");
        return major_num;
    }

    printk(KERN_INFO "Character driver registered with major number: %d & minor number: %d\n", major_num,0);
    return 0;
}

static void Character_Device_Driver_exit(void){
    printk(KERN_ALERT "Inside the %s Function\n",__FUNCTION__);
    /*Unregister the char device drv*/
    unregister_chrdev(511,"Character_Driver");
}


module_init(Character_Device_Driver_init);
module_exit(Character_Device_Driver_exit);
```

Makefile

```
Makefile
Makefile

obj-m := Character_Device_Driver.o

```

The Makefile for the character device driver is a script that automates the compilation and building of the driver code (*Character_Device_Driver.c)* into a loadable kernel module (.ko file) .

Makefile simplifies the development workflow by automating the repetitive and complex steps involved in building kernel modules. This approach is crucial for testing and deploying the driver, as it streamlines the process, eliminates potential errors from manual commands, and allows for efficient iteration during development.

```
shifana@duplesis:~/Desktop/S5$ cd char_driver
shifana@duplesis:~/Desktop/S5/char_driver$ ls
Character_Device_Driver.c  Makefile  test.c
shifana@duplesis:~/Desktop/S5/char_driver$ make -C /lib/modules/$(uname -r)/build M=$PWD modules
make: Entering directory '/usr/src/linux-headers-6.8.0-47-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/shifana/Desktop/S5/char_driver/Character_Device_Driver.o
  MODPOST /home/shifana/Desktop/S5/char_driver/Module.symvers
  CC [M]  /home/shifana/Desktop/S5/char_driver/Character_Device_Driver.mod.o
  LD [M]  /home/shifana/Desktop/S5/char_driver/Character_Device_Driver.ko
  BTF [M] /home/shifana/Desktop/S5/char_driver/Character_Device_Driver.ko
Skipping BTF generation for /home/shifana/Desktop/S5/char_driver/Character_Device_Driver.ko due to unavailability of vmlinux
make: Leaving directory '/usr/src/linux-headers-6.8.0-47-generic'
shifana@duplesis:~/Desktop/S5/char_driver$ ls
Character_Device_Driver.c    Character_Device_Driver.mod.c  Makefile        test.c
Character_Device_Driver.ko   Character_Device_Driver.mod.o  modules.order
Character_Device_Driver.mod  Character_Device_Driver.o      Module.symvers
shifana@duplesis:~/Desktop/S5/char_driver$
```

```
shifana@duplesis:~/Desktop/S5/char_driver$ sudo insmod Character_Device_Driver.ko kernel_version=6,8,0
[sudo] password for shifana:
shifana@duplesis:~/Desktop/S5/char_driver$
```

2. Driver Insertion:

Upon successful insertion, the driver should print the assigned major and minor numbers in the kernel log (this can be checked using the dmesg command).

```
shifana@duplesis:~/Desktop/S5/char_driver$ sudo tail -f /var/log/syslog
Nov  1 10:41:52 duplesis avahi-daemon[506]: avahi_key_new() failed.
Nov  1 10:42:05 duplesis kernel: [  379.631356] Character_Device_Driver: loading out-of-tree module taints kernel.
Nov  1 10:42:05 duplesis kernel: [  379.631361] Character_Device_Driver: module verification failed: signature and/or required key missing - tainting kernel
Nov  1 10:42:05 duplesis kernel: [  379.631732] kernal : 6,8,0
Nov  1 10:42:05 duplesis kernel: [  379.631736] Character driver registered with major number: 511 & minor number: 0
```

3. Device Read/Write Operations:
After insertion, you should write <FIRSTNAME>_<ROLLNO> to the device and read from it in two different ways:

1. using the ‘echo’ command for writing and the ‘cat’ command for reading.
eg:
> echo "RAMESH_B220007CS" > /dev/<device_name>
> cat /dev/<device_name>
RAMESH_B220007CS

2. using a user program written in C or any other language.
Whenever the read and write functions of the driver are called, appropriate
messages should be printed in the kernel log (e.g., Read function called! Or Write function called!).

- Creating Device Node using mknod command:

```
shifana@duplesis:~/Desktop/S5/char_driver$ sudo mknod -m 666 /dev/simple_char_dev c 511 0
shifana@duplesis:~/Desktop/S5/char_driver$
```

The mknod command is used to create a character device file (/dev/simple_char_dev) with read and write permissions.
This file serves as an interface, linking the device file to the driver with specified major (511) and minor (0) numbers. This setup allows user-space programs to interact directly with the device through the file.

- Listing all the devices:

```
shifana@duplesis:~/Desktop/S5/char_driver$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 81 video4linux
 89 i2c
 90 mtd
 99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
202 cpu/msr
204 ttyMAX
216 rfcomm
226 drm
234 aux
235 media
236 cec
237 lirc
238 mei
239 nvme-generic
240 nvme
241 hidraw
242 ttyDBC
243 bsg
244 watchdog
245 remoteproc
246 ptp
247 pps
```

```
248 rtc
249 dma_heap
250 dax
251 dimmctl
252 ndctl
253 tpm
254 gpiochip
261 accel
511 Character_Driver

Block devices:
  7 loop
  8 sd
  9 md
 11 sr
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
252 device-mapper
253 virtblk
254 mdp
259 blkext
```

- Write into device

```
shifana@duplesis:~/Desktop/S5/char_driver$ echo "SHIFANA_B221204CS" > /dev/simple_char_dev
shifana@duplesis:~/Desktop/S5/char_driver$
```

Log:

```
[ 4093.589432] Device opened 1'th times
[ 4093.589454] Write function called!
[ 4093.589456] Received from user: SHIFANA_B221204CS

[ 4093.589474] Device closed
```

- Read into device

```
shifana@duplesis:~/Desktop/S5/char_driver$ cat /dev/simple_char_dev
SHIFANA_B221204CS
```

Log:

```
[ 4120.072057] Device opened 2'th times
[ 4120.072080] Read function called!
[ 4120.072086] Sent to user: SHIFANA_B221204CS

[ 4120.072097] Read function called!
[ 4120.072117] Device closed
```

user program in C

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#define DEVICE_PATH "/dev/simple_char_dev"

int main() {
    int fd;
    char write_buffer[] = "SHIFANA_B221204CS";
    char read_buffer[1024];
    // Open the device file
    fd = open(DEVICE_PATH, O_RDWR);
    if (fd < 0) {
        perror("Failed to open the device");
        return EXIT_FAILURE;
    }
    // Write to the device
    if (write(fd, write_buffer, strlen(write_buffer)) < 0) {
        perror("Failed to write to the device");
        close(fd);
        return EXIT_FAILURE;
    }
    printf("Written to the device: %s\n", write_buffer);
    // Read from the device
    if (read(fd, read_buffer, sizeof(read_buffer)) < 0) {
        perror("Failed to read from the device");
        close(fd);
        return EXIT_FAILURE;
    }
    printf("Read from the device: %s\n", read_buffer);
    // Close the device
    close(fd);
    return EXIT_SUCCESS;
}
```

- Running the test user program:

```
shifana@duplesis:~/Desktop/S5/char_driver$ gedit test.c
shifana@duplesis:~/Desktop/S5/char_driver$ gcc ./test.c
shifana@duplesis:~/Desktop/S5/char_driver$ ./a.out
Written to the device: SHIFANA_B221204CS
Read from the device: SHIFANA_B221204CS
```

Log:

```
[ 4152.793241] Device opened 3'th times
[ 4152.793251] Write function called!
[ 4152.793253] Received from user: SHIFANA_B221204CS
[ 4152.793316] Read function called!
[ 4152.793318] Sent to user: SHIFANA_B221204CS
[ 4152.793334] Device closed
```

- Removing the driver module using rmmod:

```
shifana@duplesis:~/Desktop/S5/char_driver$ sudo rmmod Character_Device_Driver
```

Log:

```
[ 4180.029764] Inside the Character_Device_Driver_exit Function
shifana@duplesis:~/Desktop/S5/char_driver$
```

➔ Methodology

The assignment requires the development of a Linux Character Device Driver with a kernel version check at insertion.
The driver takes a kernel_version parameter, verifying it against the current kernel version before it can be loaded.
Upon successful insertion, the driver logs its assigned major and minor numbers.
It supports read and write operations, enabling interaction through commands like echo and cat.
Each read/write operation generates corresponding log entries in the kernel log.
This assignment focuses on understanding kernel module parameters, file operations, and fundamental inter-process communication in Linux.