PROBLEM STATEMENT:

The objective of this assignment is to create a character device driver with the following functionalities:

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.

Driver Insertion:

Upon successful insertion, the driver should print the assigned major and minor numbers in the kernel log

Device read/write Operations:

After insertion, write <FIRST_NAME>_<ROLLNO> to the device and read from it in two different ways. One way is using echo command from the terminal and other way is by writing a user program in C programming language.

METHODOLOGY:

Module Setup and Initialization:

- Includes necessary kernel headers (e.g., module.h, kernel.h, fs.h, uaccess.h, cdev.h).
- Defines the device name (DEVICE_NAME) and a buffer size for storing data.
- Declares key variables, including a buffer (device_buffer) to store data, major and minor numbers for device registration, and a cdev structure for representing the character device.
- Uses module_param_array to accept a kernel version array as a module parameter, making it accessible when loading the module.

File Operations:

- Implements standard file operation functions: device_open, device_release, device_read, and potentially device_write.
- Defines an operations structure (file_operations) that maps the open, release, read, and write functions to the device.

Device Registration:

- Registers the character device in the __init function using register_chrdev_region and initializes it with cdev_init.Calls cdev_add to register the cdev structure with the system, linking it to the device file operations.

Module Exit:

- Implements a __exit function that unregisters the device and cleans up resources when the module is removed.

Detailed Explanation:

Global Variables and Module parameters:

This code defines a simple character device driver for the Linux kernel. It provides basic read and write capabilities, with structured device management and logging to interact with user-space applications. Each section of the code is essential for implementing and registering the device with the Linux kernel.

```
static int major;
static int minor = 0;
static char device_buffer[BUFFER_SIZE] = {0};
static int kernel_version[3] = {0, 0, 0};
static dev_t dev_number;
static struct cdev my_cdev;
module_param_array(kernel_version, int, NULL, S_IRUGO);
MODULE_PARM_DESC(kernel_version, "Kernel version array (major, minor, patch)");
```

- **major and minor:** Variables for storing the major and minor device numbers, respectively.
- **device_buffer:** A static buffer array, device_buffer[BUFFER_SIZE], is created with a size of 256 bytes to hold device data temporarily.
- **kernel_version**: This integer array [3] holds version data passed from user space.
- **dev_number:** Stores the device number, combining both the major and minor numbers.
- **my_cdev**: A cdev structure used for managing and registering the character device.
- **module_param_array** function allows the kernel_version array to be specified as a module parameter. The S_IRUGO flag makes the parameter readable in the /sys filesystem.
- **MODULE_PARM_DESC** provides a description of the parameter.

File Operation Functions:

These functions define the core behaviour of the character device.

```
static int device_open(struct inode *inode, struct file *file) {
    printk(KERN_INFO "Device opened\n");
    return 0;
}

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

- device_open:
    - Called when the device is opened. This function typically initializes resources or counters related to the device access.
    - Logs "Device opened" informational message to the kernel log using printk.
    - Returns 0 to indicate a successful operation.

- device_release:
    - Called when the device is closed.
    - Logs "Device closed" to the kernel log with printk to indicate the device was closed.
    - Returns 0 for successful release.

```c
static ssize_t device_read(struct file *file, char __user *user_buffer, size_t length, loff_t *offset) {
    ssize_t bytes_read = 0;
    printk(KERN_INFO "Read function called\n");

    if (*offset >= BUFFER_SIZE)
        return 0;

    if (*offset + length > BUFFER_SIZE)
        length = BUFFER_SIZE - *offset;

    bytes_read = length - copy_to_user(user_buffer, device_buffer + *offset, length);
    *offset += bytes_read;
    return bytes_read;
}

static ssize_t device_write(struct file *file, const char __user *user_buffer, size_t length, loff_t *offset) {
    ssize_t bytes_written = 0;
    printk(KERN_INFO "Write function called\n");

    if (length > BUFFER_SIZE - 1)
        length = BUFFER_SIZE - 1;

    bytes_written = length - copy_from_user(device_buffer, user_buffer, length);
    device_buffer[length] = '\0';
    return bytes_written;
}
```

- device_read:
    - Allows user-space applications to read from the device buffer.
    - Parameters:
        - user_buffer: The destination buffer in user space.
        - length: The number of bytes to read.
        - offset: Position within the device's buffer to read from.
        - file: Pointer to the file structure associated with the device.
    - This function uses copy_to_user to safely copy data from the device's buffer to user_buffer.
    - Return: The number of bytes successfully read, or an error code if the read fails.

- device_write:
    - Allows user-space applications to write to the device buffer.
    - Uses copy_from_user to safely copy data from user_buffer to device_buffer.
    - Logs an informational message each time data is written.

File Operations Structure:

```
static struct file_operations oops = {
    .owner = THIS_MODULE,
    .open = device_open,
    .release = device_release,
    .read = device_read,
    .write = device_write,
};
```

- The fops structure binds the device's core functions to specific file operations (open, read, write, release).
- This structure includes pointers to the functions, allowing the kernel to invoke them when performing file operations on the device.
- This structure is essential because it lets the kernel know which function to call when these operations are invoked on the device.

Device Registration:

```
static int __init Intialize(void) {
int current_ver[3] = {
        (LINUX_VERSION_CODE >> 16) & 0xFF,
        (LINUX_VERSION_CODE >> 8) & 0xFF,
        LINUX_VERSION_CODE & 0xFF
    };

    if (kernel_version[0] != current_ver[0] || kernel_version[1] != current_ver[1] || kernel_version[2] != current_ver[2]) {
        printk(KERN_ERR "Kernel version mismatch: Expected %d.%d.%d, got %d.%d.%d\n",
               current_ver[0], current_ver[1], current_ver[2],
               kernel_version[0], kernel_version[1], kernel_version[2]);
        return -EINVAL;
    }

    if (alloc_chrdev_region(&dev_number, minor, 1, DEVICE_NAME) < 0) {
        printk(KERN_ALERT "Failed to allocate major and minor numbers\n");
        return -1;
    }

    major = MAJOR(dev_number);
    minor = MINOR(dev_number);

    printk(KERN_INFO "Device registered: %s with Major number %d, Minor number %d\n", DEVICE_NAME, major, minor);

    cdev_init(&my_cdev, &oops);
    if (cdev_add(&my_cdev, dev_number, 1) < 0) {
        unregister_chrdev_region(dev_number, 1);
        return -1;
    }
    return 0;
}
```

The __init function is responsible for initializing and registering the device with the system:

- alloc_chrdev_region(&dev_number, minor, 1, DEVICE_NAME): Registers a device number, storing the major number.

- cdev_init(&my_cdev, &fops): nitializes the cdev structure with pointers to the file_operations.
- cdev_add(&my_cdev, dev_number, 1): Registers the device with the kernel.

Module Exit:

```
static void __exit Exit(void) {
    cdev_del(&my_cdev);
    unregister_chrdev(major, DEVICE_NAME);
    printk(KERN_INFO "Device unregistered\n");
}
```

The __exit function unregisters the device upon module removal:

- unregister_chrdev_region(dev_number, 1): Frees up the major number, releasing it back to the system.

Intialization of init and exit macros:

```
module_init(Intialize);
module_exit(Exit);
```

module_init:

- Specifies that char_driver_init is the initialization function that should be called when the module is loaded.
- This function sets up the device, checks kernel version compatibility and registers the device with the system.

module_exit:

- Specifies that char_driver_exit is the cleanup function that should be called when the module is unloaded. This function unregisters the device and logs its removal

Testing the functionality of the driver:

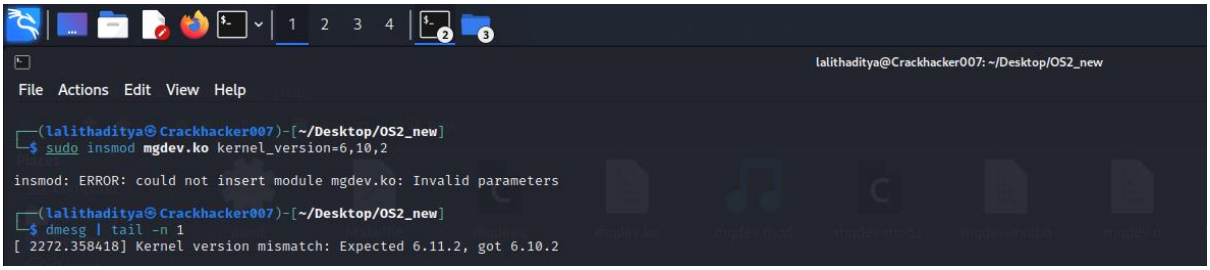
1. **Kernel Version Check**

   The kernel version used to compile the module is:

```
┌──(lalithaditya㉿Crackhacker007)-[~/Desktop/OS2]
└─$ uname -v

#1 SMP PREEMPT_DYNAMIC Kali 6.11.2-1kali1 (2024-10-15)
```

And clearly the driver can't be inserted if provided the wrong kernel version (Refer below screenshot). We can also see log when we first try to insert the driver using kernel version 6.10.2 which gave us an error as seen in screenshot.



The driver should only be inserted if the provided kernel version matches the version used to compile the module. Which is clearly shown in below screenshot.

```
┌──(lalithaditya㉿Crackhacker007)-[~/Desktop/OS2_new]
└─$ sudo insmod mgdev.ko kernel_version=6,11,2

┌──(lalithaditya㉿Crackhacker007)-[~/Desktop/OS2_new]
└─$ dmesg | tail -n 10
```

2. **Driver Insertion**:

   Upon successful insertion, the driver should print the assigned major and minor numbers in the kernel log. Which is shown in below screenshot.

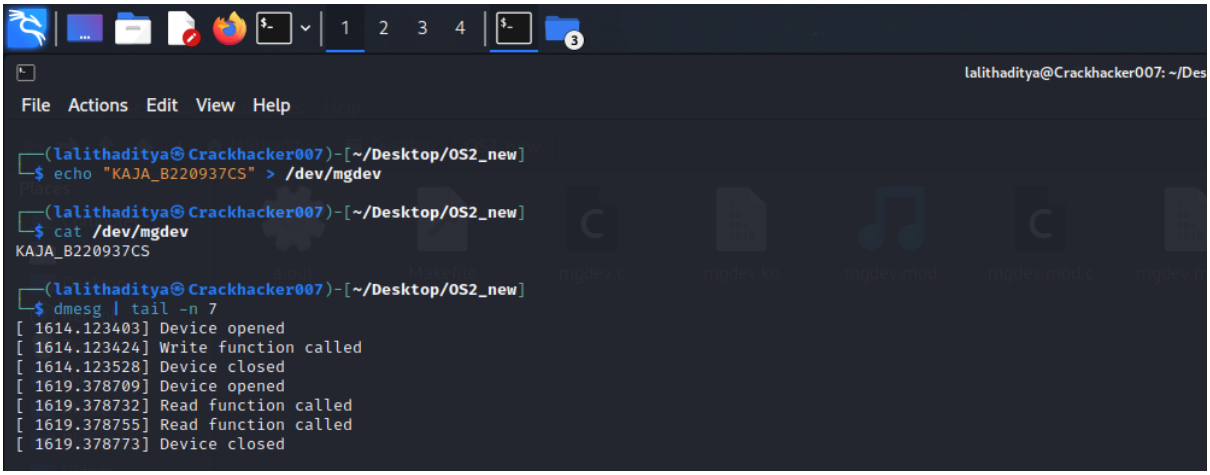
```
┌──(lalithaditya㉿Crackhacker007)-[~/Desktop/OS2_new]
└─$ sudo insmod mgdev.ko kernel_version=6,11,2

┌──(lalithaditya㉿Crackhacker007)-[~/Desktop/OS2_new]
└─$ dmesg | tail -n 10
[  614.611710] [drm:vmw_msg_ioctl [vmwgfx]] *ERROR* Failed to open channel.
[  690.864812] mgdev: loading out-of-tree module taints kernel.
[  690.864825] mgdev: module verification failed: signature and/or required key missing - tainting kernel
[  690.869587] Device registered: mgdev with Major number 245, Minor number 0
[  738.557928] Device unregistered
[  758.056950] Device registered: mgdev with Major number 244, Minor number 0
[  768.845686] Device unregistered
[  797.026920] Device registered: mgdev with Major number 243, Minor number 0
[  813.266862] Device unregistered
[  862.598600] Device registered: mgdev with Major number 242, Minor number 0
```

   From the last kernel log the major number is 242 and minor number is 0.

3. **Device read/write operations**:

   a. using the 'echo' command for writing and the 'cat' command for reading: we can clearly see from the kernel log the file operations performed.



b. using a user program “test.c” written in C. The program is given below:

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define DEVICE_PATH "/dev/mgdev"

int main() {
    int fd;
    char *message = "KAJA_B220937CS";
    char buffer[100];

    fd = open(DEVICE_PATH, O_RDWR);
    if (fd < 0) {
        perror("Failed to open device");
        return 1;
    }

    write(fd, message, strlen(message));

    read(fd, buffer, sizeof(buffer));

    printf("Read from device: %s\n", buffer);

    close(fd);

    return 0;
}
```

Output:

Whenever the read and write functions of the driver are called, appropriate messages should be printed in the kernel log. Which is shown in below screenshot last four logs .