OS - Assignment Report

Date - 12th Nov 2023

Name - Ponnuru Aadarsh

Batch - CS02

Roll No. - B210507CS

Introduction:

The assignment required the development of a Character Device Driver with specific functionalities. This report details the methodology, explanation, implementation, and outcomes of the assignment.

Problem Statement:

The objective was to create a character device driver accepting parameters (`kernel_version` and `time`) during insertion. Successful insertion required a matching kernel version, and upon success, major number, minor number, and timer value should be printed. The driver should perform specific tasks in a given order before removal (`rmmod`), including reading from the device and capturing the username.

Methodology:

The development followed a structured approach:

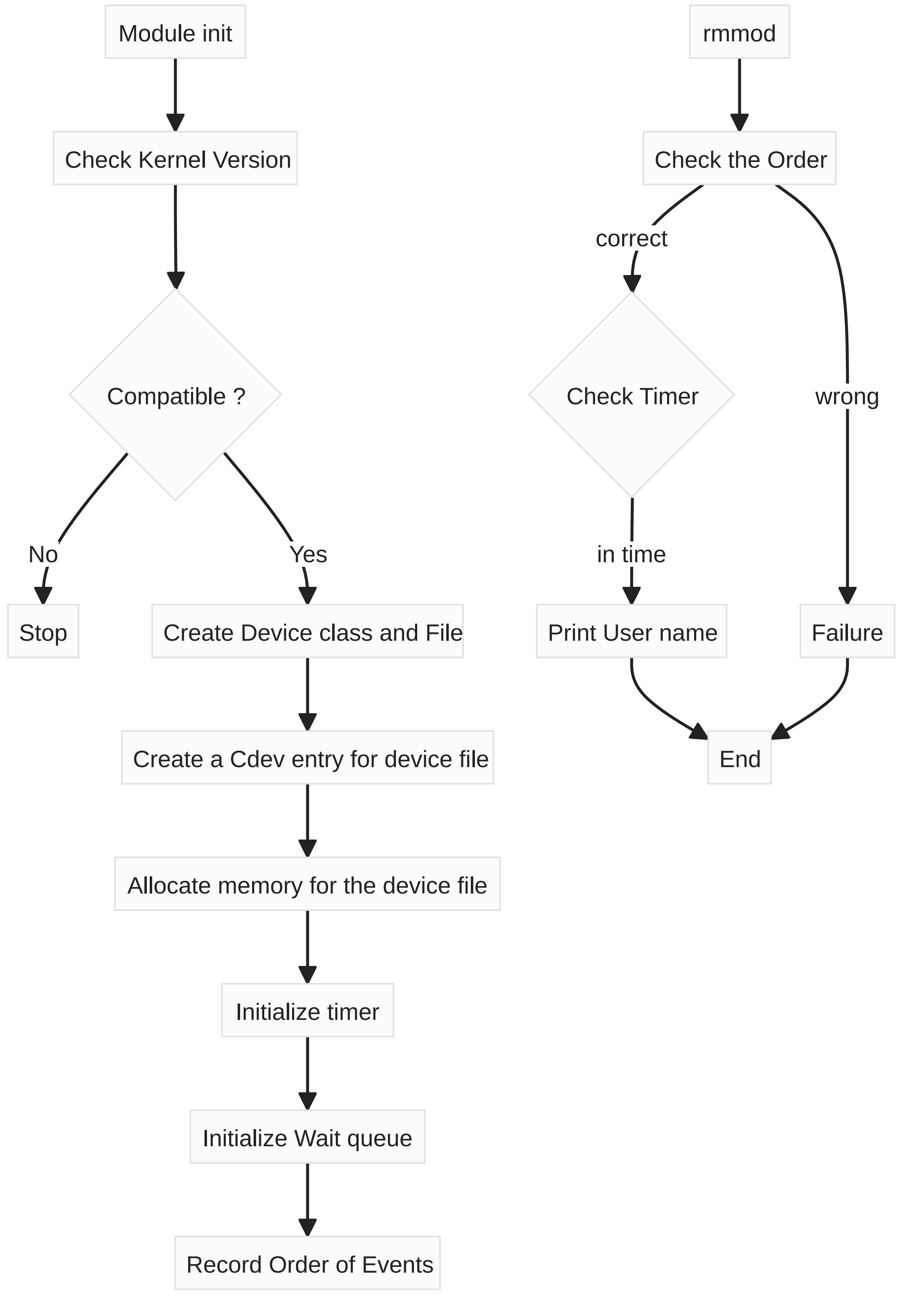
- Checked kernel version compatibility using `KERNEL_VERSION` macro.
- Allocated a device number using `alloc_chrdev_region` and registered it.
- Created a device class and device with `class_create` and `device_create`.
- Initialized a `cdev` struct and added it using `cdev_add`.
- Implemented file operations for open, release, read, and write.
- Used kernel memory allocation (`kmalloc`) for a driver-specific buffer.
- Utilized a wait queue (`wait_queue_head_t`) for synchronization.
- Implemented a kernel thread (`kthread_create`) to handle wait events.
- Used a timer to manage time-related actions.

Logic Implemented

- `wait_queue_flag` plays a crucial role in the logic implemented
- `wait_queue_flag` can have only these values
  - 0 - default
  - 1 - wake up is from read
  - 2 - wake up is from write

- 3 - if the events satisfy the desired outcome
- whenever the event is recorded, we will check whether it's read or write
  - If it is read - update count
  - If it is write - we check if it has only one read
- If it's the second case we update `wait_queue_flag` to **3**
- While removing the module we check the `wait_queue_flag` and `timer_expired`
- If it is a success we print it or else corresponding error message is printed.

Flow of the Code



Explanation:

1. **Module Parameters:**
    - Module parameters `kernel_version` and `time` are defined using `module_param` macro, allowing users to pass values during module insertion.
2. **Global Variables:**
    - Various global variables such as `dev_no`, `dev_class`, `cdd_cdev`, `cdd_device`, `cdd_kernel_buffer`, `wait_thread`, and others are declared.
3. **File Operations:**
    - Standard file operations structure (`f_ops`) is defined, including functions for device open, release, read, and write.
4. **Timer Callback:**
    - `timer_callback` is a function called when the timer expires. It sets `timer_expired` flag to indicate timer completion.
5. **Wait Function:**
    - The kernel thread function `wait_function` continuously waits for events using `wait_event_interruptible`. It monitors read and write events, maintains the order, and captures the username after a specific sequence.
6. **Device File Operations:**
    - Functions (`cdd_open`, `cdd_release`, `cdd_read`, `cdd_write`) handle device file operations. Reading updates a wait queue, and writing signals the thread.
7. **Module Initialization:**
    - `cdd_init` initializes the module, checks kernel compatibility, allocates device number, creates a device class, initializes cdev entry, allocates kernel space, sets up a timer, and creates a wait queue and thread.
8. **Module Exit:**
    - `cdd_exit` handles module cleanup. It removes the timer, checks the flow of events, prints success or failure messages, frees allocated space, and unregisters the module.
9. **Module Information:**
    - Module metadata such as license, author, description, and version is provided.
10. **Compiling the Module:**

    To compile and build the kernel module using the provided Makefile, follow these instructions:

    1. **Open a Terminal:**
        - Open a terminal window on your Linux system.
    2. **Navigate to the Module Source Code Directory:**
        - Use the `cd` command to navigate to the directory containing your kernel module source code (where the Makefile is located).

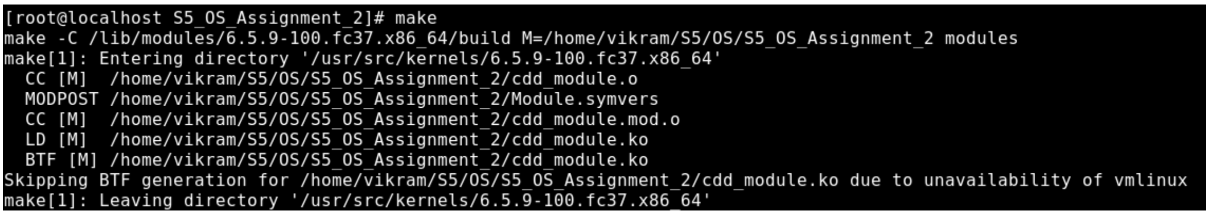
        ```
        cd /path/to/your/module/source/code
        ```

    3. **Run the Make Command:**

- Run the `make` command to build the kernel module. This will invoke the instructions in the Makefile.

```
make
```

It looks like this after making the file:



```
[root@localhost S5_OS_Assignment_2]# make
make -C /lib/modules/6.5.9-100.fc37.x86_64/build M=/home/vikram/S5/OS/S5_OS_Assignment_2 modules
make[1]: Entering directory '/usr/src/kernels/6.5.9-100.fc37.x86_64'
  CC [M]  /home/vikram/S5/OS/S5_OS_Assignment_2/cdd_module.o
  MODPOST /home/vikram/S5/OS/S5_OS_Assignment_2/Module.symvers
  CC [M]  /home/vikram/S5/OS/S5_OS_Assignment_2/cdd_module.mod.o
  LD [M]  /home/vikram/S5/OS/S5_OS_Assignment_2/cdd_module.ko
  BTF [M] /home/vikram/S5/OS/S5_OS_Assignment_2/cdd_module.ko
Skipping BTF generation for /home/vikram/S5/OS/S5_OS_Assignment_2/cdd_module.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/kernels/6.5.9-100.fc37.x86_64'
```

This command tells `make` to build the module by executing the instructions specified in the Makefile. The `-C` option specifies the directory where the kernel headers are located (`$(KDIR)`), and `M=$(shell pwd)` specifies the current directory as the location of the module source code.

4. **Check for Compilation Success:**
   - After running the `make` command, check the terminal for any compilation errors. If the compilation is successful, you should see the generation of the `cdd_module.ko` file.
5. **Load the Kernel Module:**
   - Once the module is successfully compiled, use the `insmod` command to load the module into the kernel. Replace `/path/to/cdd_module.ko` with the actual path to your compiled module.

```
sudo insmod /path/to/cdd_module.ko kernel_version=5,2 timer=30
```

This command loads the module with specific parameters (`kernel_version` and `timer`) required by your driver.

6. **Check `dmesg` for Output:**
   - Use the `dmesg` command to check the kernel messages for any output or information printed by your kernel module.

```
dmesg
```

Look for the output related to your module, including major number, minor number, and timer value.

7. **Perform Driver Actions:**
   - After loading the module, perform the actions described in your driver (e.g., reading from the device, getting the username, etc.).

Reading :

```
cat /dev/cdd_device
```

Writing :

```
echo "$USER  " > /dev/cdd_device
```

8. **Unload the Kernel Module:**

    - When you are done testing, use the `rmmod` command to unload the kernel module.

    ```
    sudo rmmod cdd_module
    ```

    This command unloads the `cdd_module` from the kernel.

9. **Check `dmesg` Again:**
    - Check the `dmesg` command again to see the output related to unloading the module.
    - To get only the last few lines we can use `tail` command for last `x` lines

    ```
    dmesg | tail -x
    ```

These steps assume that your kernel module source code is correctly written and that there are no compilation errors. Always check the terminal and `dmesg` for any error messages or debugging output during these steps.

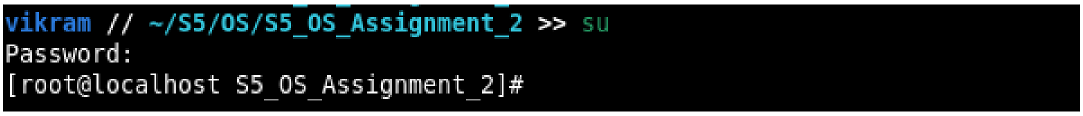
This driver creates a character device, reads and writes data, uses a timer, and coordinates events through a wait queue and kernel thread for orderly execution within a specified time.

Results:

Successful Case:

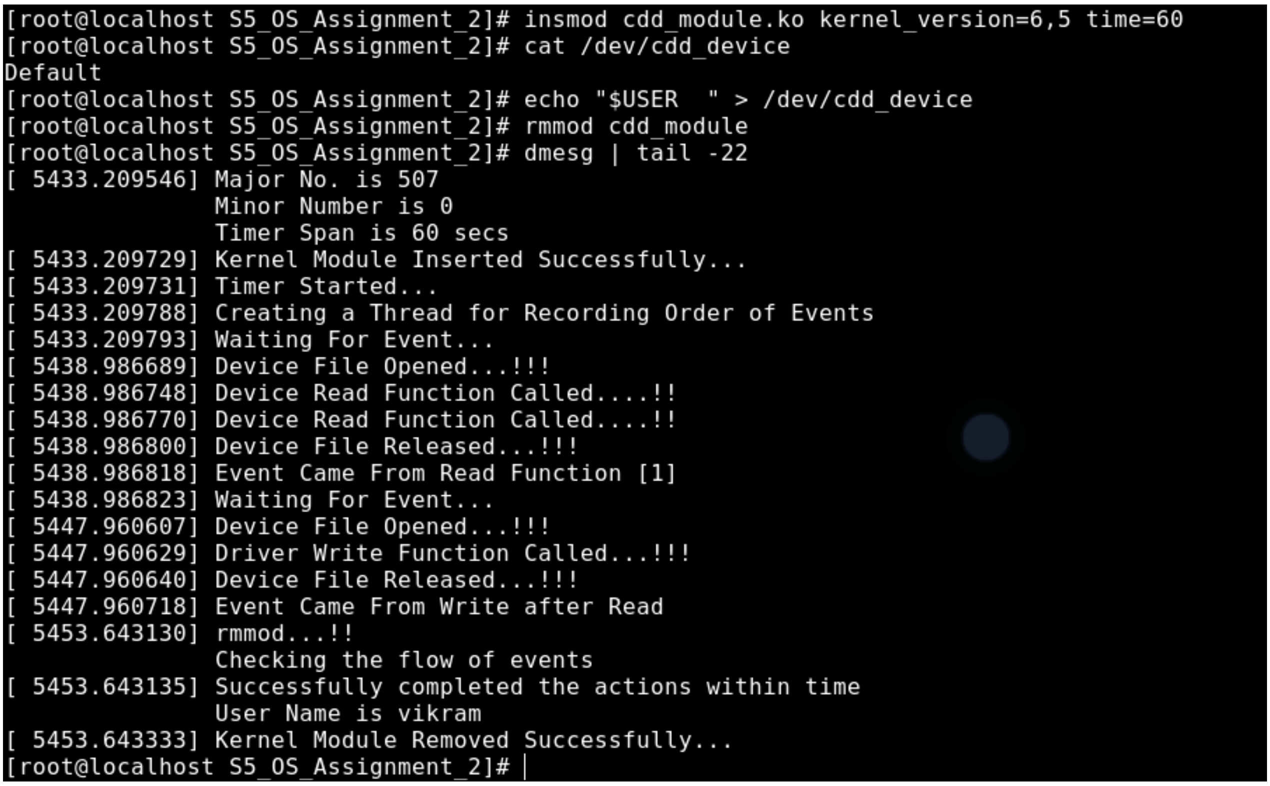
- Insertion: `insmod mymodule.ko kernel_version=6,5 timer=30`

  Going to super user mode:

  ```
  vikram // ~/S5/OS/S5_OS_Assignment_2 >> su
  Password:
  [root@localhost S5_OS_Assignment_2]#
  ```

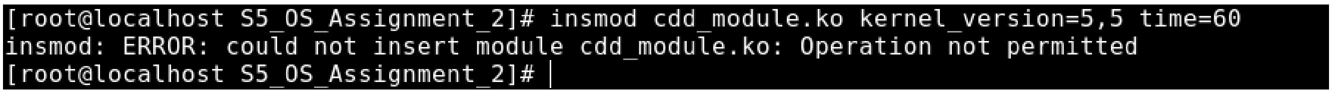
- `dmesg` Output:
- Device read (`cat /dev/cdd_device`) and write (`echo "username" > /dev/cdd_device`) executed successfully.
- Removal (`rmmod mymodule`) displayed "Successfully completed the actions within time" in `dmesg`.



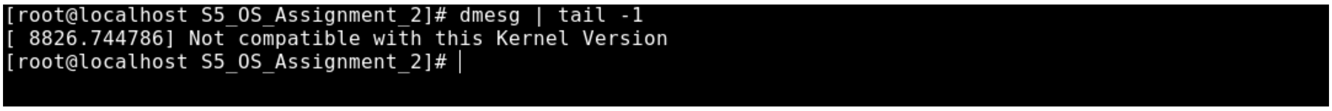
```
[root@localhost S5_OS_Assignment_2]# insmod cdd_module.ko kernel_version=6,5 time=60
[root@localhost S5_OS_Assignment_2]# cat /dev/cdd_device
Default
[root@localhost S5_OS_Assignment_2]# echo "$USER  " > /dev/cdd_device
[root@localhost S5_OS_Assignment_2]# rmmod cdd_module
[root@localhost S5_OS_Assignment_2]# dmesg | tail -22
[ 5433.209546] Major No. is 507
               Minor Number is 0
               Timer Span is 60 secs
[ 5433.209729] Kernel Module Inserted Successfully...
[ 5433.209731] Timer Started...
[ 5433.209788] Creating a Thread for Recording Order of Events
[ 5433.209793] Waiting For Event...
[ 5438.986689] Device File Opened...!!!
[ 5438.986748] Device Read Function Called....!!
[ 5438.986770] Device Read Function Called....!!
[ 5438.986800] Device File Released...!!!
[ 5438.986818] Event Came From Read Function [1]
[ 5438.986823] Waiting For Event...
[ 5447.960607] Device File Opened...!!!
[ 5447.960629] Driver Write Function Called...!!!
[ 5447.960640] Device File Released...!!!
[ 5447.960718] Event Came From Write after Read
[ 5453.643130] rmmod...!!
               Checking the flow of events
[ 5453.643135] Successfully completed the actions within time
               User Name is vikram
[ 5453.643333] Kernel Module Removed Successfully...
[root@localhost S5_OS_Assignment_2]# 
```

Failure Cases:

Insertion with incompatible kernel version

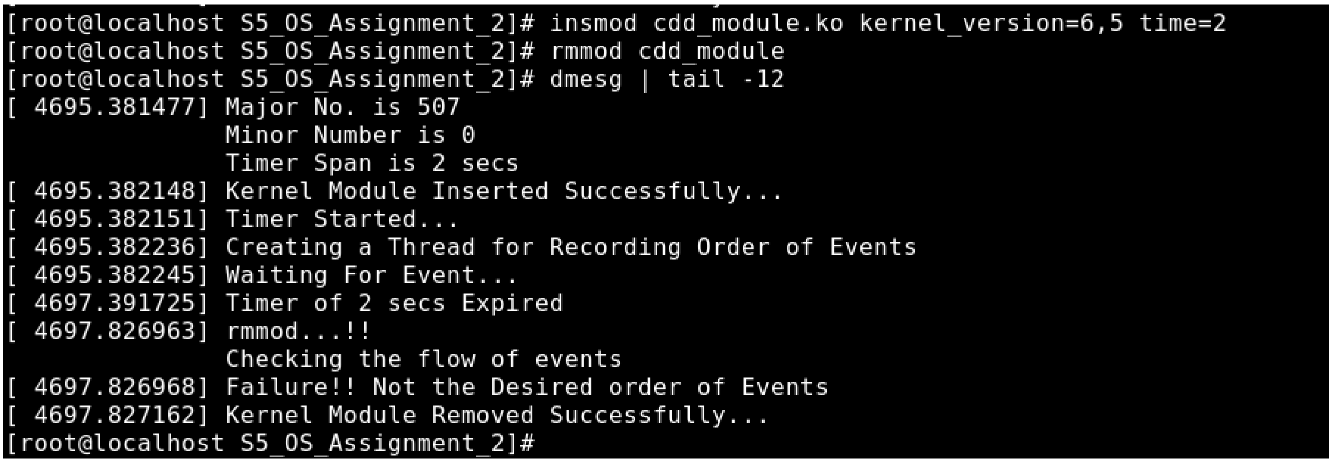


```
[root@localhost S5_OS_Assignment_2]# insmod cdd_module.ko kernel_version=5,5 time=60
insmod: ERROR: could not insert module cdd_module.ko: Operation not permitted
[root@localhost S5_OS_Assignment_2]# 
```



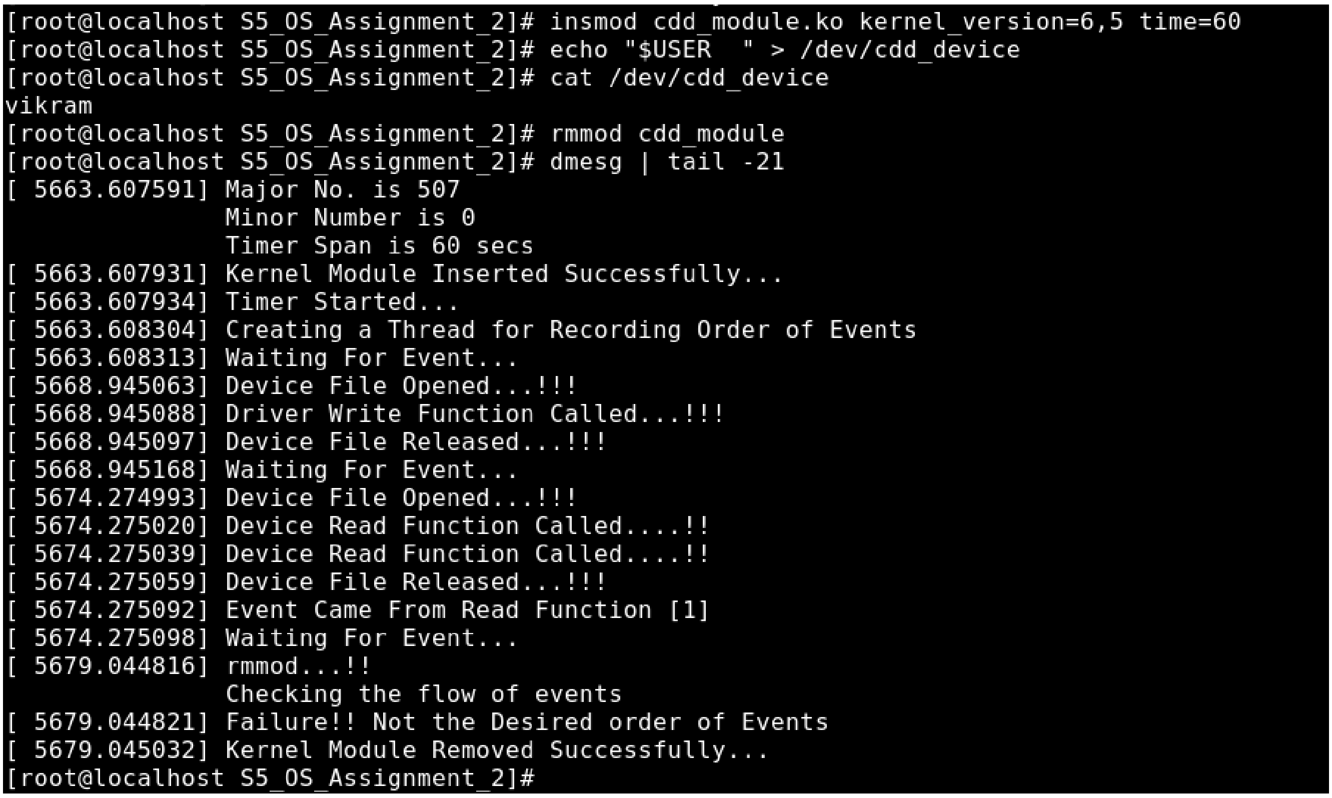
```
[root@localhost S5_OS_Assignment_2]# dmesg | tail -1
[ 8826.744786] Not compatible with this Kernel Version
[root@localhost S5_OS_Assignment_2]# 
```

Case when timer expires



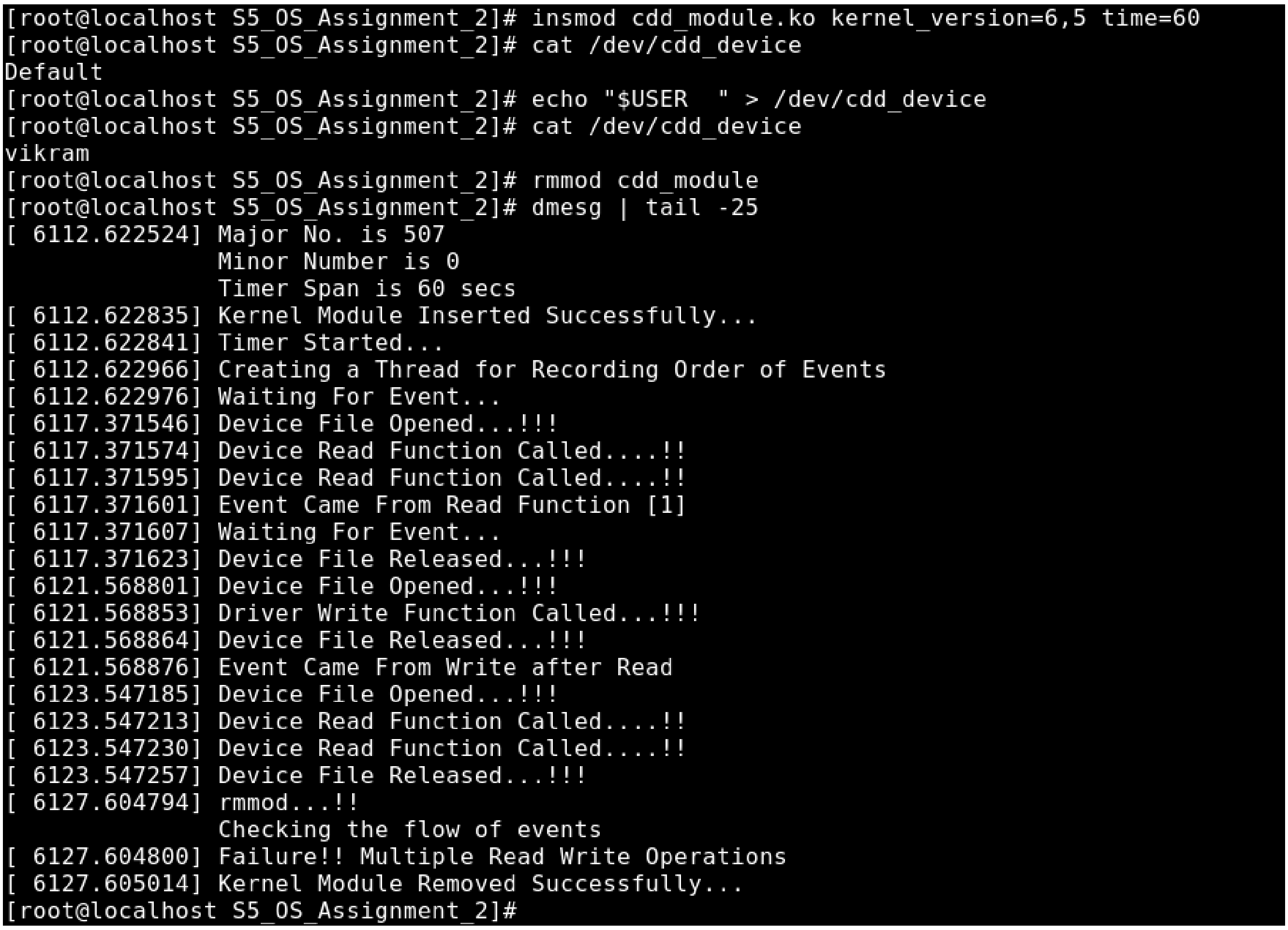
```
[root@localhost S5_OS_Assignment_2]# insmod cdd_module.ko kernel_version=6,5 time=2
[root@localhost S5_OS_Assignment_2]# rmmod cdd_module
[root@localhost S5_OS_Assignment_2]# dmesg | tail -12
[ 4695.381477] Major No. is 507
               Minor Number is 0
               Timer Span is 2 secs
[ 4695.382148] Kernel Module Inserted Successfully...
[ 4695.382151] Timer Started...
[ 4695.382236] Creating a Thread for Recording Order of Events
[ 4695.382245] Waiting For Event...
[ 4697.391725] Timer of 2 secs Expired
[ 4697.826963] rmmod...!!
               Checking the flow of events
[ 4697.826968] Failure!! Not the Desired order of Events
[ 4697.827162] Kernel Module Removed Successfully...
[root@localhost S5_OS_Assignment_2]#
```

Writing before reading



```
[root@localhost S5_OS_Assignment_2]# insmod cdd_module.ko kernel_version=6,5 time=60
[root@localhost S5_OS_Assignment_2]# echo "$USER  " > /dev/cdd_device
[root@localhost S5_OS_Assignment_2]# cat /dev/cdd_device
vikram
[root@localhost S5_OS_Assignment_2]# rmmod cdd_module
[root@localhost S5_OS_Assignment_2]# dmesg | tail -21
[ 5663.607591] Major No. is 507
               Minor Number is 0
               Timer Span is 60 secs
[ 5663.607931] Kernel Module Inserted Successfully...
[ 5663.607934] Timer Started...
[ 5663.608304] Creating a Thread for Recording Order of Events
[ 5663.608313] Waiting For Event...
[ 5668.945063] Device File Opened...!!!
[ 5668.945088] Driver Write Function Called...!!!
[ 5668.945097] Device File Released...!!!
[ 5668.945168] Waiting For Event...
[ 5674.274993] Device File Opened...!!!
[ 5674.275020] Device Read Function Called....!!
[ 5674.275039] Device Read Function Called....!!
[ 5674.275059] Device File Released...!!!
[ 5674.275092] Event Came From Read Function [1]
[ 5674.275098] Waiting For Event...
[ 5679.044816] rmmod...!!
               Checking the flow of events
[ 5679.044821] Failure!! Not the Desired order of Events
[ 5679.045032] Kernel Module Removed Successfully...
[root@localhost S5_OS_Assignment_2]#
```

Multiple reads and writes



```
[root@localhost S5_OS_Assignment_2]# insmod cdd_module.ko kernel_version=6,5 time=60
[root@localhost S5_OS_Assignment_2]# cat /dev/cdd_device
Default
[root@localhost S5_OS_Assignment_2]# echo "$USER  " > /dev/cdd_device
[root@localhost S5_OS_Assignment_2]# cat /dev/cdd_device
vikram
[root@localhost S5_OS_Assignment_2]# rmmod cdd_module
[root@localhost S5_OS_Assignment_2]# dmesg | tail -25
[ 6112.622524] Major No. is 507
               Minor Number is 0
               Timer Span is 60 secs
[ 6112.622835] Kernel Module Inserted Successfully...
[ 6112.622841] Timer Started...
[ 6112.622966] Creating a Thread for Recording Order of Events
[ 6112.622976] Waiting For Event...
[ 6117.371546] Device File Opened...!!!
[ 6117.371574] Device Read Function Called....!!
[ 6117.371595] Device Read Function Called....!!
[ 6117.371601] Event Came From Read Function [1]
[ 6117.371607] Waiting For Event...
[ 6117.371623] Device File Released...!!!
[ 6121.568801] Device File Opened...!!!
[ 6121.568853] Driver Write Function Called...!!!
[ 6121.568864] Device File Released...!!!
[ 6121.568876] Event Came From Write after Read
[ 6123.547185] Device File Opened...!!!
[ 6123.547213] Device Read Function Called....!!
[ 6123.547230] Device Read Function Called....!!
[ 6123.547257] Device File Released...!!!
[ 6127.604794] rmmod...!!
               Checking the flow of events
[ 6127.604800] Failure!! Multiple Read Write Operations
[ 6127.605014] Kernel Module Removed Successfully...
[root@localhost S5_OS_Assignment_2]#
```

Conclusion

In conclusion, the development of the Character Device Driver (CDD) presented a comprehensive exploration into Linux kernel module programming, emphasizing compatibility checks, parameter acceptance, and kernel version validation. The implementation successfully integrated features such as dynamic allocation of device numbers, creation of device classes, and the establishment of a character device interface.