OPERATING SYSTEMS ASSIGNMENT - 2

Submitted by

Nandana Anand

B220424CS

Problem Statement

The objective of this assignment is to create a character device driver in Linux with specific functionalities, including kernel version checking, dynamic driver registration, and read/write operations. The following functionalities are to be implemented:

1. Kernel Version Check:
   The driver should accept a kernel version(as an array parameter which specifies the current kernel version) as a module parameter and only load successfully if the specified version matches the version used to compile the module.
2. Driver Insertion:
   The driver should dynamically assign device numbers and log the assigned major and minor numbers in the kernel log upon successful insertion(which can be checked using the dmesg command).
3. Device Read/Write Operations:
   It should support both read and write operations as follows to write <FIRSTNAME>_<ROLLNO> to the device and read from it in two different ways:
   1. Direct shell commands (echo and cat)
   2. Using a custom user program written in C or any other language.

The driver will output relevant messages to the kernel log during read/write operations, providing insights into each operation.

Methodology

Define Kernel Version Check: Configure the driver to accept a kernel version parameter. During initialization, verify the module's compatibility with the running kernel version.

Driver Setup: Register a character device, allocate memory for a buffer, and set up device functions for handling read and write operations.

Implement Read/Write Operations: Add read and write functions to manage data transfer between user space and the device, with logging for each operation.

User Interaction: Demonstrate driver functionality using echo and cat commands and a user-space program to read and write data.

Cleanup: On module exit, free resources and unregister the device.

Detailed Explanation

The following steps have been undertaken for implementing:

Kernel Version Check

1. The driver accepts an array parameter called *kernel_version* using the *module_param_array* macro. This parameter is passed at the insertion stage(see README for steps to insert the module) and specifies the required kernel version.
2. Inside the *os_driver_init* initialization function, the driver converts this array to a format compatible with the *LINUX_VERSION_CODE*.
3. The version is then checked against *LINUX_VERSION_CODE* to verify compatibility. If there is a mismatch, an error message is printed and the driver terminates initialization with -*EINVAL,* preventing further loading.

Driver Initialization and Device Setup

1. The *alloc_chrdev_region* function is used to dynamically allocate major and minor numbers for the device. These numbers are required to identify the device file in the system.
2. The *cdev_init* function initialises the *cdev* structure and associates it with the *file_operations* structure, which contains pointers to the device functions for opening, reading, writing and releasing the device.
3. Using *cdev_add,* the device is registered with the kernel. If this step fails, the code is designed to clean up and release any resources acquired so far.
4. The device class is created using *class_create* and the actual device file is created using *device_create* allowing user-space applications to interact with the driver through */dev/os_driver_device*

Buffer Allocation

1. The driver allocates a buffer of size *BUFFER_SIZE* using *kmalloc,* which will be used to store data written to the device. The buffer is allocated during initialization and freed during driver removal to avoid memory leaks.

Device File Operations

1. The file operations(read,write,open and release) are defined in the *fops* structure.
2. Open: The *os_device_open* function is called when the device file is opened. A message is printed in the kernel log to indicate that the device has been opened.
3. Read: The *os_device_read* function is responsible for reading data from *device_buffer* to the user-space buffer provided. It checks for end-of-file conditions and uses *copy_to_user* to transfer data.
4. Write: The *os_device_write* function writeds data from the user-space buffer into *device_buffer.* If the incoming data size exceeds *BUFFER_SIZE,* only the first *BUFFER_SIZE* bytes are written to prevent buffer overflow.

5. Release: The *os_device_release* function is called when the device is closed and it prints a message to the kernel log.

Verification of Read and Write Operations

After inserting the driver, data can be written to the device using *echo* and read using *cat* from the terminal. Additionally, a custom user program can interact with the device using standard file operations in C. The kernel log(viewed using *dmesg*) provides messages for each read and write operations, which serve as verification points for the functionality.

Driver Removal

During module removal, the *os_driver_exit* function performs cleanup operations to free any allocated resources. The buffer is freed, the device file and class are destroyed, and the device numbers are unregistered. This ensures that the driver is removed from the system cleanly.

Verification

1. After running make, insert the driver by running the command(check README for detailed instructions on running):

   sudo insmod os_driver.ko kernel_version=6,1,112

   Note: Here, the kernel version (6.1.112) has been found after running the uname -a command:

   ```
   nandana@nandana-hplaptop:~/OS_A2/src$ uname -a
   Linux nandana-hplaptop 6.1.0-26-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.112-1 (2024-09-30) x86_64 GNU/Linux
   ```

2. Run sudo dmesg to view the log messages. If there were no errors, you can see these messages:

   ```
   [15646.605439] Kernel version matches. Proceeding with module insertion.
   [15646.609265] Kernel Module Successfully Inserted..
   [15646.609273] MajorNo=237      MinorNo=0
   ```

   We can observe the insertion was successful.

3. Get general information of the module using `sudo modinfo os_driver.ko`

```
filename:       /home/nandana/OS_A2/src/os_driver.ko
description:    Character device driver for OS assignment 2
author:         Nandana
license:        GPL
depends:
retpoline:      Y
name:           os_driver
vermagic:       6.1.0-26-amd64 SMP preempt mod_unload modversions
parm:           kernel_version:array of int
```

4. Run sudo lsmod to see the list of all currently loaded kernel modules. You can notice the module ‘os_driver’ loaded:

```
nandana@nandana-hplaptop:~/OS_A2/src$ sudo lsmod
Module                  Size  Used by
os_driver              16384  0
ccm                    20480  0
rfcomm                 94208  4
snd_seq_dummy          16384  0
snd_hrtimer            16384  1
```

5. Use the command `cat /proc/devices` to check the major numbers assigned to various devices. Output:

```
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 81 video4linux
 99 ppdev
116 alsa
128 ptm
136 pts
153 spi
180 usb
189 usb_device
202 cpu/msr
216 rfcomm
226 drm
237 os_driver
```

6. In /sys/module/<driver_name>/parameters, one can find a list of configurable parameters specific to a loaded kernel module (driver). We can see kernel_version variable listed when we run ls /sys/module/os_driver/parameters

```
nandana@nandana-hplaptop:/$ ls /sys/module/os_driver/parameters
kernel_version
```

7. Check for device files in the /dev directory. Run ls -l /dev

```
crw-------  1 root    root    237,     0 Nov  1 17:39 os_driver_device
```

8. We can check if the read and write are happening as required.

   Using commands: For running the following commands, one might need sudo access. Run sudo -s and type in the password which will now allow us to run commands with sudo permissions.

   1. echo “NANDANA_B220424CS” > /dev/os_driver_device
      Run dmesg to get the log messages:

      ```
      [18829.735511] Driver Open Function Called...!!!
      [18829.735540] Driver Write Function Called...!!!
      [18829.735558] Driver Release Function Called...!!!
      ```

   2. cat /dev/os_driver_device

      ```
      root@nandana-hplaptop:/home/nandana# cat /dev/os_driver_device
      NANDANA_B220424CS
      ```

      Run dmesg to get the log messages:

      ```
      [18922.191907] Driver Open Function Called...!!!
      [18922.191951] Driver Read Function Called...!!!
      [18922.191969] Driver Read Function Called...!!!
      [18922.191999] Driver Release Function Called...!!!
      ```

      The read function is called twice because cat or similar programs often perform multiple read operations until they reach the end of the file (EOF).

   Using C program:

   1. Compile the program: The user program as of now directly writes NANDANA_B220424CS (<FIRSTNAME>_<ROLLNO>) into the device. But it can be modified to take input from the user through the terminal and then write it into the device. This can be implemented by removing the following comments in the code:

      ```
      22      // // Prompt the user for input
      23      // printf("Write to the device: ");
      24      // if (fgets(write_buf, sizeof(write_buf), stdin) == NULL) {
      25      //     perror("Failed to read input");
      26      //     close(fd);
      27      //     return EXIT_FAILURE;
      28      // }
      29      // // Remove the newline character from input if it exists
      30      // write_buf[strcspn(write_buf, "\n")] = '\0';
      31
      ```

```c
int main() {
    int fd;
    // char write_buf[256];
    char write_buf[256] = "NANDANA_B220424CS";
    char read_buf[256] = {0};

```

Also remove the comments for write_buf and comment out the 2nd write_buf line.

If running the actual user.c code, the following output can be obtained.

- gcc user.c
- Run sudo ./a.out

```
nandana@nandana-hplaptop:~/OS_A2/src$ gcc user.c
nandana@nandana-hplaptop:~/OS_A2/src$ sudo ./a.out
Wrote to the device: NANDANA_B220424CS
Read from the device: NANDANA_B220424CS
```

- Run dmesg to see the log messages

```
[19453.337082] Driver Open Function Called...!!!
[19462.584570] Driver Write Function Called...!!!
[19462.584600] Driver Read Function Called...!!!
[19462.584633] Driver Release Function Called...!!!
```

9. For removing the driver module, run the sudo rmmod os_driver and check the log messages

```
[20684.691707] Kernel Module Removed Successfully...
```