DM510 ASSIGNMENT 2

Kernel Module

Esra Duran (esdur23)

Completed this assignment with Bruno Campos Uribe (brcam23) and Tuğra Demirel (tudem23)

APRIL 16, 2023

Table of Contents

Introduction

In this assignment we are asked to create a module for kernel. In this module we need 2 devices which are communication tools between user and the system. As the program requires, those devices need to have a relationship with 2 different buffers (Buffer 0 and Buffer 1). When the process writes to the first device (/dev/dm510-0) , the written data is stored in the buffer 1. If that buffer is full, the system needs to wait until the other process of the buffer 1 which is reading from /dev/dm510-1. Similarly, if the system wants to read from the device /dev/dm510-0, reading will be conducted trough the bounded buffer 0, if there is nothing to read, the process will be in a waiting mode.

Design Decisions

Firstly, we need to create a different structure besides buffer, to define devices. Device has pointers of buffers (write_buffer and read_buffer). Those buffers represent the specific buffers for each devices. The buffer itself has also different properties such as buffersize (this can be determined by the user at first, or we can simply use the default version) , writing points and reading points (that shows the starting point of the text in that buffer). The buffer structure also have the actual buffers which are pointed to the buffer space (this is the place that we write or read from).

The buffer allocation is made in the "dm510_init_module" function. Also, we first finding the amount of spaces to write into. In the writing and the cleaning process, the buffers are freed when there is not exist any reader or writer in the system. It prevents us from using unnecessary memory, we make the system more efficient.

Implementation

1-Inıt

In this function devices are created with using predetermined major and minor numbers. We start by registering the devices for the driver. We allocate space for the devices by using their sizes and the number of devices by using kmalloc. After having the devices, the buffer allocations are done (buffer 0 and buffer 1). Also, in order to satisfy the thread processing, we added queues to buffers (write buffer and read buffer). In the initialization step, we used setup_cdev function. If those initializations fails, we simply free the devices.

2-Clean Up

This process is for freeing the spaces that are allocated for buffers and also devices. When the buffers are freed, the cdevs are also deleted. Then, we conclude the process by unregistering the devices.

3-Open

In the opening phase, we have two options, one of them is read mode and the other one is write mode. In the reading mode, we check the number of readers and allow only one reader. In the reading mode, we again check the number of the writers and be sure that it does not exceed the maximum number of writers. If they are not satisfied, we return -EBUSY error.

4-Release

In this function, we get the device that is in use of private data. If the process is in read mode, the number of readers of the device will be decremented by 1 unit. If the process is in the writing mode, similarly, the number of writers of the device will be decremented. We used atomic_dec() and atomic_inc() function for those processes. After those steps, the buffers unlocks the mutex, so that if there is no writers or readers in the system, device can allow new ones to come in a process. This release function simply arranges the queue and frees the buffer if there is no process going on.

5-Read

The read function first attempts to acquire a lock on the read buffer associated with the device linked to the file pointer. If the read pointer and write pointer are equal, indicating there is no new data to read, the function will either wait until new data is available or return an error.

When the reading is in the process, the user can read the specified amount of text, or till the end. This is done by copying the data from the buffer to the user's space. If this copying step fails, we unlock the mutex and return an error. When the reading is completed successfully, any writers waiting on the reader's buffer are woken up as the buffer is no longer full.

6-Write

Similar to read process, the write function also starts with acquiring a lock on the write buffer of the related device. Then, we check whether there is any free by using spacefree function. If there exists enough space, we simply get the count (amount of area we can use for writing) with using the end point of the buffer and writing point. If the writing pointer and the reader pointer are the same, the buffer is empty.

If we have enough free space, we copy the text from the user to the buffer and increment the writing point by the size of that text (remained space, if the space is less than the text size).Then, If it reaches to the end of the allocated space, the writing point will be equal to the starting of that buffer.

Lastly, we unlocked the device and woke it up in case it was on the sleeping mode because of no writing process.

7-ioctl

In this function, users can decide on what the buffer size or maximum number of readers will be, and they can set them to any number. This usage provides user convenience.

Processes

Before initializing the driver, we initialized the queues for the devices. Those queues are needed in case of there are sleeping or interrupts in the system. Sleeping mode is used in the blocking/nonblocking activities. We used mutex_unlock() and mutex_lock() functions to arrange the interruptibility. Readers will be in the sleeping mode when the read_buffer is empty and also when the max number of readers is achieved. Writers will be also in the sleeping mode when there is no free space to write

Test

In the testing part, we have a basic module test which firstly writes to a device, and the other device tries to read that. After that, the same processes are going on but in the opposite direction. We see that the two devices are succeeded to read and write the correct data. The expectations and the actual results is compared in order to detect any problem in the driver.

We conducted another tests by thinking of more readers and more writers than we expected in the system. As we know, the reading process can be made by many readers at the same time, while the writing process is made only at a time. When we have much more than readers or writers, we cannot open the device and cannot perform the desired task.

Conclusion

In this assignment, we try to understand the basics of the driver and device allocation structure. We tried to conduct basic test, which are passed. But still, there may be leaks so that the driver cannot pass. In order to detect those, we need to implement the sleeping, thread, blocking I/O, or invalidities. This assignment overall reads and writes the data trough the devices using corresponding buffers and obey the rules of sleeping, blocking and interruptions.