ASSIGNMENT 2

REPORT

Prepared by

Team Number: 17

Hadif Yassin Hameed	**B190513CS**
Abhinav Ajithkumar	**B190461CS**
Hanna Nechikkadan	**B190420CS**
Faseem Shanavas	**B190515CS**
Harimurali J	**B190547CS**

Course: CS3003D: Operating Systems

Date: 24/10/2021

Problem Statement

Create a simple device driver (character device) for the compiled kernel of the previous Assignment and test it with a sample application.

Methodology

1. Write a kernel module to create the character device and its associated file and to implement its i/o functions.
    a. Register the device, dynamically assigning the Major number.
    b. Create a device file for the registered device.
    c. Implement `open`, `close`, `read`, `write` functions for the device.
2. Build the kernel module and insert it into the kernel.
3. Build and run the test application to interact with the character device.

Explanation

Introduction

The aim of this endeavour was to create and test a simple character device driver on the previously compiled linux kernel version 5.14.1. This driver implements primitive I/O functions, and we can test this functionality with the help of a sample application.

Definitions

Device Driver : It's a collection of files that allow one or more hardware devices to connect with the operating system of a computer.Without drivers, the computer would be unable to send and receive data to hardware devices correctly.
They are critical for a computer system's correct operation because without a device driver, the hardware fails to function as intended, i.e., it fails to perform the function or action for which it was designed.

Character devices : They are devices that do not have physically addressable storage media and perform I/O in a byte stream. Some common examples include taped drives and storage ports. In UNIX , devices are represented with the help of a special artefact called a `device file`, which is stored in the `/dev` directory.

Steps Taken To Solve The Problem

Writing the Kernel module

File operations: The `file_operations` structure is defined in `linux/fs.h`, and holds pointers to functions defined by the driver that perform various operations on the device. Each field of the structure corresponds to the address of some function defined by the driver to handle a requested operation. `open`, `release`, `read` and `write` file operations are implemented for the device.

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

`open` handler: for opening the device (allocating resources).
`release` handler: for closing the device (releasing resources).
`read` handler: for reading data from the device (copying data from kernel buffer to user buffer). `read` also clears the kernel buffer.
`write` handler: for writing data to the device (copying data from user buffer to kernel buffer). New data is appended to the kernel buffer. If the buffer is full, write fails until a read is issued to clear the buffer.

Registering the device: Character devices are accessed through device files, usually located in `/dev`. The major number tells you which driver handles which device file. The minor number is used only by the driver itself to differentiate which device it's operating on, just in case the driver handles more than one device. Adding a driver to your system means registering it with the kernel. This is synonymous with assigning it a major number during the module's initialization. You do this by using the `register_chrdev` function, defined by `linux/fs.h`.

```
Major = register_chrdev(0, DEVICE_NAME, &fops);
```

If you pass a major number of 0 to register_chrdev, the return value will be the

dynamically allocated major number. Since we do not know the major number in advance, a device file is created with minor number 0 within the driver using the `device_create` function of the linux kernel.

Unregistering the device: We cannot allow root to remove the kernel module whenever it pleases. If a process opens the device file and then the kernel module is removed, using the file will result in a call to the memory location where the appropriate function (`read`/`write`) was previously located. If we're lucky, no other code was loaded in that location, and we'll get a dreadful error message. If we're unlucky, another kernel module was loaded into the same location, resulting in a jump into another kernel function. It would be impossible to predict the outcomes, but they couldn't be good.
When we remove the kernel module using `rmmod` command, the `cleanup_module` function in the device driver removes the device file and unregisters the char device.

```
void cleanup_module(void)
{
       device_destroy(pClass, devNo);      // Remove the /dev/chardev file
       class_destroy(pClass);              // Remove class /sys/class/chardev
       unregister_chrdev(Major, DEVICE_NAME);
       printk(KERN_INFO "Unregistered char device %s with major %d.\n", DEVICE_NAME, Major);
}
```

Build the Kernel module

We create a Makefile to build the kernel module. The Makefile should generate a kernel object. For convenience, we use the same Makefile to compile the application program.

```
obj-m+=chardev.o

all:
   make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
   $(CC) test.c -o test
clean:
   make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
   rm test
```

We can then build the kernel using the `make` command.

```
~/Documents/OS_Theory/chrdev make
make -C /lib/modules/5.14.1-MANJARO/build/ M=/home/batman/Documents/OS_Theory/chrdev modules
make[1]: Entering directory '/home/batman/linux-5.14.1'
  CC [M]  /home/batman/Documents/OS_Theory/chrdev/chardev.o
  MODPOST /home/batman/Documents/OS_Theory/chrdev/Module.symvers
  CC [M]  /home/batman/Documents/OS_Theory/chrdev/chardev.mod.o
  LD [M]  /home/batman/Documents/OS_Theory/chrdev/chardev.ko
make[1]: Leaving directory '/home/batman/linux-5.14.1'
cc test.c -o test
```

Load the Kernel module

The compiled kernel module can be dynamically loaded using the `insmod` command.

```
~/Documents/OS_Theory/chrdev sudo insmod chardev.ko
[sudo] password for batman:
~/Documents/OS_Theory/chrdev lsmod | grep chardev
chardev                16384  0
~/Documents/OS_Theory/chrdev cat /proc/devices | grep chardev
509 chardev
~/Documents/OS_Theory/chrdev ls -l /dev | grep chardev
crw-------  1 root   root   509,     0 Oct 24 19:20 chardev
```

The `init_module` function is called when the module is loaded. We can look for the module entry using `lsmod` and the device entry in `/proc/devices` and the newly created `/dev/chardev` file to verify the same. We can also see messages from the kernel module in the syslog.

```
Oct 24 15:16:13 batjaro kernel: Registered char device chardev with major 509.
```

Test the Kernel module

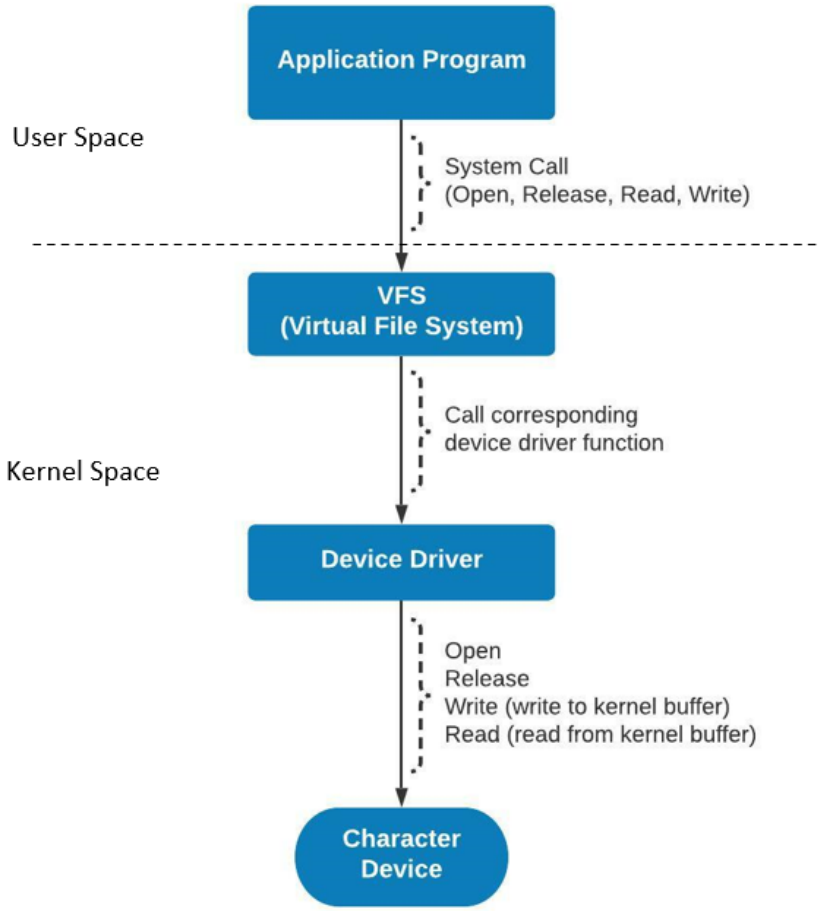
The application program to test the device driver is compiled along with the kernel module. To test the device driver run the test program as root. The user application infinitely takes input from the user and writes it to the device on `ENTER` or when the buffer is full. On `$` input it writes the string to the and reads from the device.

```
~/Documents/OS_Theory/chrdev sudo ./test
Enter message to send to the character device:
A string is sent to the device on ENTER or when buffer is full.
A read is triggered on '$' key.
Reading from the device also clears its kernel buffer
You can close the application using keyboard interrupt (ctrl+C).
Message to send to the character device:
[55 chars left]: hello from the other side
[21 chars left]: i must've called a thousand times
[9 chars left]: to tell youu
[3 chars left]: sorry
[1 chars left]: d
Buffer full...
Press any key to read back from the device...
Reading from the device...
The received message is:
hello from the other side
i must've called a thousand times
to tell you
sorry
d
Message to send to the character device:
[80 chars left]:
```

```
Oct 24 17:18:42 batjaro kernel: File opened 1 times
Oct 24 17:18:48 batjaro kernel: Current Buffer: hello from the other side
                                 (26 bytes)
Oct 24 17:18:57 batjaro kernel: Current Buffer: hello from the other side
                                i must've called a thousand times
                                 (60 bytes)
Oct 24 17:19:00 batjaro kernel: Current Buffer: hello from the other side
                                i must've called a thousand times
                                to tell you
                                 (72 bytes)
Oct 24 17:19:02 batjaro kernel: Current Buffer: hello from the other side
                                i must've called a thousand times
                                to tell you
                                sorry
                                 (78 bytes)
Oct 24 17:19:08 batjaro kernel: Current Buffer: hello from the other side
                                i must've called a thousand times
                                to tell you
                                sorry
                                d (79 bytes)
Oct 24 17:19:08 batjaro kernel: Read 79 bytes.
Oct 24 17:19:51 batjaro kernel: File closed
```

Hence we have verified that our character device driver works as specified.

API Diagram



References

1. https://tldp.org/LDP/lkmpg/2.6/html/x569.html
   The Linux Kernel Module Programming Guide
2. https://docs.oracle.com/cd/E19683-01/806-5222/6je8fjvep/index.html
   Drivers for Character Devices by Oracle
3. http://www.tutorialsdaddy.com/linux-device-drivers/writing-simple-character-device-driver/
   Writing a simple Character Device Driver , 16th August 2017
4. https://www.slideshare.net/VandanaSalve/introduction-to-char-device-driver
   Introduction to char device driver by Vandana Salve, 2nd April 2013