COL331 OS Assignment 3 - Report

Hard Track

Aryan Dua - 2020CS50475

Instructions to run the module:

1. Make
2. Sudo insmod mychardev.ko
3. Compile the writer code and execute
4. Compile the reader code and execute
5. Sudo rmmod mychardev.ko

Functionality:

1. I made a character device driver for 2 devices. The major number is being allocated dynamically and the minor numbers are 1 for the writer and 0 for the reader.
2. I maintain the 1MB storage using a global variable called data_device.
3. There are 4 file operations taken care of - open, close, read and write. The main functionality of the LIFO devices is in the read and write functions. The open and close functions were just to maintain the structural integrity.
4. I maintain 2 pointers, 1 for the current read index and 1 for the current write index.
5. Whenever write() is called, the write index is incremented by the number of bytes copied into the character device's memory
6. Whenever read() is called, the read index is incremented by the number of bytes sent to the user.
7. To maintain the LIFO structure, I reverse the string when read() is called and then copy the data. For reversing, I made my own custom function.
8. The module starts with the init function and closes with the cleanup function.
9. This is where we register and unregister the character devices.

Design Decisions:

1. I had initially implemented a reader device that just reads, but later modified it so that after it reads n characters, it starts reading from the next character the next time it is called.
2. My EOF file is $. Reading from an empty file returns this character,

Extra:

1. I have added spinlocks before and after the copyfromuser() in the write() function
2. I have used work_queues to call the read() function if it was originally called while a write was happening. What it will do is read the current status of the memory at the time it is called(status before the spinlock) and then it will call it again.
3. I had earlier implemented this using timers and making a callback function, but work_queues turned out to be more efficient.
4. This prevents multiple users from writing into the file simultaneously. When a user is writing, the others can only see the status of the memory either before the write started, or after it finished. (blocking)
5. I developed the module as an LKM but then inserted it into the kernel

Problems faced:

1. There was a Wframe error that says that the stack memory allocated to my function is greater than the default 1024 bytes. To take care of this I added a line in the MakeFile changing the maximum stack size to 1MB.
2. It was difficult to get the indexes right, more often than not, the reader was reading other sections of the memory, which were not modified by the writer.
3. The kernel just hung when I tried doing sudo insmod mychardev.ko at a very random moment. It said that I don't have the permission to create a device. This happened at the very last hour. So, I submitted the latest version of the working code I had.

Testing:

Writer.c:

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

int main (int argc , char * argv [])
{
   char device [100];
   char user_msg [100];

   strcpy ( device , argv [1]) ;
   strcpy ( user_msg , argv [2]);

   // device name /dev/mychardev-1
   int fd = open(device , O_WRONLY ) ;
   printf("fd = %d\n", fd);

   printf("writing %s\n", user_msg);
   write(fd, user_msg, strlen(user_msg));

}
```

Reader.c

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

int main (int argc , char * argv [])
{
   char device [100];
   char user_msg [100];

   strcpy ( device , argv [1]) ;

   // device name /dev/mychardev-0
   int fd = open(device , O_RDONLY ) ;
```

```
    printf("Reading\n");
    read(fd, user_msg, 16);
    printf("String read is %s\n", user_msg);

}
```

Sources Referred:

1. https://sysprog21.github.io/lkmpg/
2. https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html
3. https://stackoverflow.com/questions/17744754/step-to-build-a-built-in-kernel-module