**Assignment #1** – Linux Kernel Programming

**Part #1** – Hello World Kernel Module

In general, Part #1 was very entertaining to code and implement. I utilized several online resources to solve the coding assignment including YouTube and the following kernel module programming guide: https://tldp.org/LDP/lkmpg/2.4/html/book1.htm**.** My coding solution simply sets up the basic parameters of the module (license, author, etc.) and implements the corresponding functions module\_init() and module\_exit() that are to be used when creating my module. It was very interesting to learn about the kernel and its differences compared to userspace code. The use of printk() to print messages to the kernel log along with the unique KERN\_INFO macros were quite interesting to learn about and use since they differed so greatly from standard C programming. I had several challenges in completing this part of the assignment. Firstly, it was very difficult to get my virtual machine set up properly for kernel development. I tried several options including QEMU, Oracle’s VirtualBox, and VMware of which the last one ended up working for me properly. Additionally, setting up VSCode for kernel development was quite challenging since the path couldn’t find the proper header files to be able to properly lint my implementation. I simply removed the C/C++ extension and utilized Vim or plan VSCode for the majority of the assignment. Lastly, I was confused about finally removing the module since it would take a while for the “Goodbye, world!” message to appear in the kernel log after running sudo dmesg.

**Part #2** – Memory Driver Module

Part #2 was a very interesting portion of the assignment to implement. I mainly utilized this YouTube series that I found describing how to create device files and write/read to them along with a couple of StackOverFlow pages. My coding solution implements the standard code needed to get a module up and running along with opening up a device file with 512kB of buffer storage so that a user can write/read messages to it accordingly. I had several challenges with this part of the assignment. The first problem that I ran into was almost crashing my VMWare Ubuntu installation due to memory errors caused by my kernel log. I ended up solving this by clearing out these logs, but I haven’t managed to figure out the error that is causing it to fill it up so quickly. Additionally, I ran into some problems with my read function, where it would continue in a loop forever causing my code to crash. Similarly, when running my user space program, it would often result in a result that stated killed. I ended up being able to solve this by fixing what I was returning in the function as well as my use of pointers.

**Analysis of Data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1 Byte Test (Write)** | **1 Byte Test (Read)** | **64 Bytes Test** | **~1k Bytes Test** | **~65k Bytes Test** | **~524k Bytes Test** |
| 0.000004 | 0.000001 | 0.000003 | 0.000012 | 0.000711 | 0.004590 |
| 0.000002 | 0.000001 | 0.000002 | 0.000012 | 0.000103 | 0.003456 |
| 0.000001 | 0.000001 | 0.000002 | 0.000019 | 0.000766 | 0.005566 |
| 0.000001 | 0.000001 | 0.000002 | 0.000013 | 0.000744 | 0.004079 |
| 0.000001 | 0.000001 | 0.000002 | 0.000012 | 0.000753 | 0.002530 |
| 0.000001 | 0.000001 | 0.000002 | 0.000070 | 0.000767 | 0.004092 |
| 0.000001 | 0.000001 | 0.000002 | 0.000013 | 0.000820 | 0.005746 |
| 0.000001 | 0.000001 | 0.000002 | 0.000014 | 0.000903 | 0.003848 |
| 0.000001 | 0.000001 | 0.000002 | 0.000013 | 0.000774 | 0.001990 |
| 0.000001 | 0.000001 | 0.000002 | 0.000012 | 0.000722 | 0.005769 |
| **Average Value & Bytes/Secs** | | | | | |
| 0.0000014 | 0.000001 | 0.0000021 | 0.000019 | 0.0007063 | 0.0041666 |
| 714285.7143 | 1000000 | 30476190.48 | 53894736.84 | 92787767.24 | 125831133.3 |

This graph is very interesting as it shows just how much they different tests differ in terms of performance. As can be seen, as we increase the amount that must be read/written, the time required increases exponentially along with the file sizes. This, of course is to be expected, however what was interesting was the increasing bytes/sec as the amount to write/read increased. There is most certainly a lack of precision which can be artificially deflating the bytes/sec calculation for the smaller sizes. Additionally, some overhead can be created in getting the system ready for the smaller sizes since it doesn’t take much time to actually read/write for different sizes. So, the time it takes for the system to set everything up is definitely playing a role in making it so that in reading/writing larger files, the bytes/sec calculation was larger.

**Additional Information**

Note: I utilized the following resource very heavily: https://tldp.org/LDP/lkmpg/2.4/html/book1.htm in the development of my kernel code. Significant parts for the implementation of the main module functions were utilized (namely, the general setup of the code).