Part Two: Memory Driver

Implementation

My solution consists of two sections: initializing and preparing the kernel module's buffer and driver, and implementing the functions the userspace needs to access. The first section consists of:

- Init
    - Space is allocated for the buffer
    - Major number, class, and device are created
- Exit
    - Everything initialized in init is safely destroyed

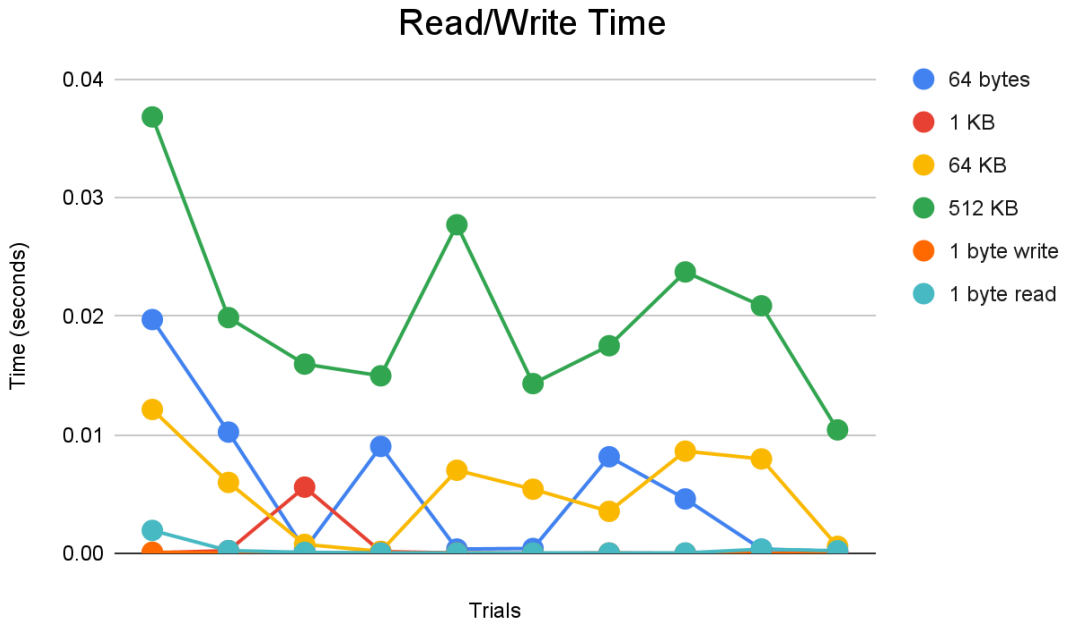
The second section consists of:

- Read
    - Checks user's parameters (offset and length) will read within the limits of the kernel space buffer
    - Copies the kernel space buffer content into the user space buffer
- Write
    - Checks user's parameters (offset and length) will write within the limits of the kernel space buffer
    - Copies the user space buffer content into the kernel space buffer
- Llseek
    - SEEK_SET: changes position to the specified user parameter by updating the file's position to offset
    - SEEK_CUR: changes position to the specified user parameter + current position by updating the file's position to offset + current position
    - SEEK_END: changes position to the specified user parameter + size of file by updating the file's position to end of file + offset

Difficulties

I struggled with understanding the concepts of class creation, device creation, and major numbers – I had no background with any of them and spent a lot of time researching these ideas. Testing was also difficult given that debugging is very different than C programs. Sudo privileges threw me off again here – I had conflicting reports from different sources of what commands to run with sudo, and I spent a lot of time debugging errors related to terminal commands rather than actual code.

Figures



Raw Data:

	Bytes Read/Written					
Trial	**64 bytes**	**1 KB**	**64 KB**	**512 KB**	**1 byte write**	**1 byte read**
1	0.019732	0.00007	0.012153	0.03681	0.000123	0.00196
2	0.010246	0.000253	0.006004	0.019896	0.000057	0.000261
3	0.000351	0.005607	0.000781	0.015977	0.000053	0.000109
4	0.009038	0.000178	0.000195	0.014991	0.000065	0.000063
5	0.000388	0.000057	0.007031	0.027724	0.000047	0.000059
6	0.000438	0.000034	0.005432	0.014334	0.000015	0.000057
7	0.008166	0.000032	0.003565	0.017526	0.000047	0.000056
8	0.004613	0.000031	0.00864	0.023746	0.000012	0.00006
9	0.000381	0.00003	0.007984	0.02089	0.00001	0.000378
10	0.000196	0.000029	0.000619	0.010433	0.00001	0.000243

Analysis

512 KB has the highest latency, which makes sense as it is the biggest buffer size to read/write. 64 bytes and 64 KB seem to oscillate for second place – surprising considering 1 KB is greater than 64 bytes. The 1 byte write and read operations are the fastest – which also makes sense considering they are the smallest. Copying data to and from user space seems to add overhead as the pattern indicates larger buffer space is correlated with higher latency. Caching effects could also be at play – cached data would result in faster speeds. The speeds for the 512 KB buffer reduce significatnly with each trial, which could be a result of caching.

AI

Chat GPT 4 (Clarity) wrote the majority of the code (90%) of the code base. I changed the switch case in llseek to have the correct offset calculations and the return values of the read and write functions. I also added lseek to the test file as this was erroring with Chat GPT's implementation. Claude.ai added checks for different offset + length checks in the read and write functions and wrote test files to check these conditions. Clarity's Chat GPT gave me a template for the Makefile and Claude helped me debug the issues – ultimately I had to modify the run section to add sudo privileges as well as alter the insmod and rmmod commands.