GH repo: https://github.com/evgerritz/cpsc429_labs/tree/1_p4
Run the Rust mymem test using ‘cargo run’ inside mymem/test_mymem
Run the Rust multithreaded mymem test using ‘cargo run’ inside mymem/threads_rust
The rust_mymem module is in samples/rust_mymem.rs

Part 4.

- Acknowledgements
    - Referenced the provided resources, using the webinar and the rust-for-linux github documentation
    - Referenced the sample miscdev driver included with Rust-for-Linux
- Overview
    - mymem/test_mymem/src/main.rs implements the mymem timing test originally written in C in part 2.
    - mymem/threads_rust/src/main.rs implements the multithreaded test program originally written in C in part 3.
    - samples/rust_mymem.rs implements the rust_mymem Rust kernel module
- Challenges
    - Getting the kernel compilation to work was a pain. I had to mess around with the EXTRAVERSION setting in the kernel Makefile until the modules had the right versionmagic string. Otherwise, the target machine would not accept the compiled modules.
    - Debugging the rust kernel module proved to be quite difficult. I kept getting segmentation faults and was not sure why they were happening. It seems that the Rust compiler was laxer regarding the kernel module code.
- Questions
    - Is there any difference in performance compared to the program written in C? Why?
        - As can be seen in the data below, the performance was quite comparable for smaller sizes, with the C program only being significantly faster for the largest transfer size tested. This makes sense since Rust has a development goal to be close to C in performance.
    - What difference does your Rust-based implementation make regarding the data race problem you had to deal with in Part 3? Does Rust compiler help? If so, how?
        - Rust helps because Rust’s ownership system prevents multiple mutable references to a single object being shared across threads. Instead, Rust forces the programmer to use the proper tools for synchronizing shared state in the form of atomic reference counters and mutexes. This meant

that as soon as I had a Rust program without any compiler errors, the program worked exactly as intended!

- Now repeat the same experiment you did at Part 3 with a large value of N (>100000) with both the test program and Memory Driver in Rust. Is there any difference in performance compared to the program written in C? Why?
  - I measured the performance of both test programs using N=150,000 and W=50. The C test program with the Rust module took a total of 2 minutes, 27 seconds of CPU time. The Rust test program with the Rust module took a total of 2 minutes, 9 seconds. This is a little surprising, since I would have expected the C program to have slightly better performance. The difference in time is small and this method of measuring time is subject to a good deal of noise caused by background tasks, so I am not sure if this result is reliable. It is possible that the C bindings generated by bindgen, which are used by the C program to interface with the module are more inefficient or somehow introduce overhead that is not present when the Rust program is interfacing with the Rust driver.
- Now that both the test module and Memory driver are implemented in safe Rust, how does the Rust compiler help solve the data race issues from Part 3?
  - Theoretically, there could still be data race issues arising from within the execution of the kernel module. However, as before, Rust ensures that these race issues cannot occur by forcing the programmer to write code that guarantees only one function has access to the shared memory allocation at a time. In this case, I used a ref and a mutex to ensure the file operations were atomic.

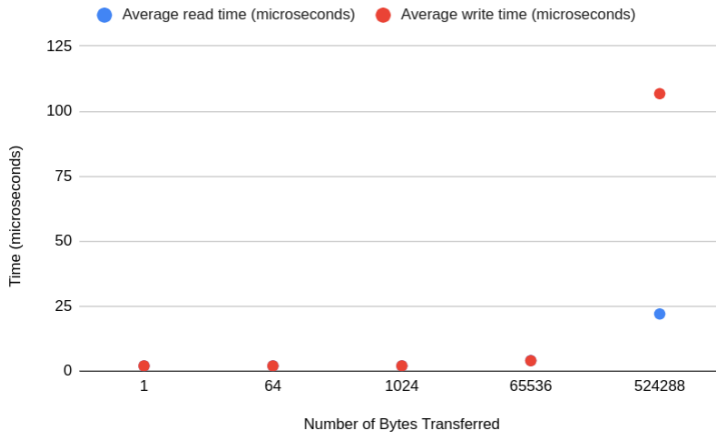
- Data/plot
  -

Size	Average Read Time (C) (n=1000) (microseconds)	Average Write Time (C) (n=1000) (microseconds)
1	2	2
64	2	2
1024	2	2
65536	3	4
524288	22	12

  -

Size	Average Read Time (Rust) (n=1000) (microseconds)	Average Write Time (Rust) (n=1000) (microseconds)
1	2.02	2.01
64	2.01	2.02

1024	2.03	2.03
65536	4.02	4.07
524288	106.72	22.01



-