Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton

Section 1: Experimental Purpose

The purpose of this experiment was two-fold. Firstly, it gave us an opportunity to further explore algorithms such as the FIFO and replacement algorithms that had been taught in class as well as allow us to create our own algorithm that performed better than the aforementioned algorithms. Secondly, it allowed us to explore the processes behind virtual memory and how that is implemented, specifically with fault handlers and how different policies behind operating system fault handlers affect performance.

The experiments that we ran used 100 pages and increments of 1 frame starting at 3 up to a maximum of 100. We wrote a Python script to automate the testing and graphing process. An example of one of the commands the test script ran is “./virtmem 100 25 fifo alpha”. We developed all of our code on Docker and used that to run initial tests. The test script uses matplotlib in Python3, which is not present in the Docker instance, so that was run on a local Linux machine.

Section 2: Explanation of Custom Page Replacement Algorithm

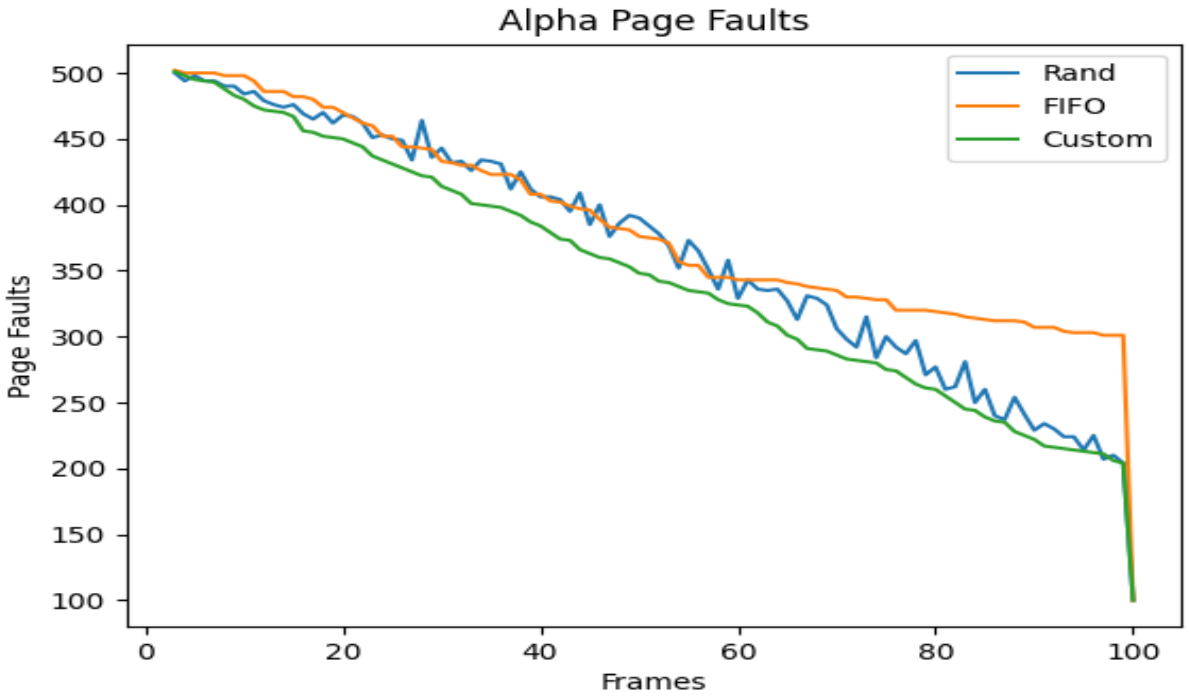
Our custom page replacement algorithm uses a modified last-in, first-out (LIFO) methodology, with a one-page buffer. The rationale behind this is that the first few numbers read into the program are likely important and will be read a multitude of times. The one-page cache is implemented in case there are situations where two of the later-read pages need to be used at the same time.

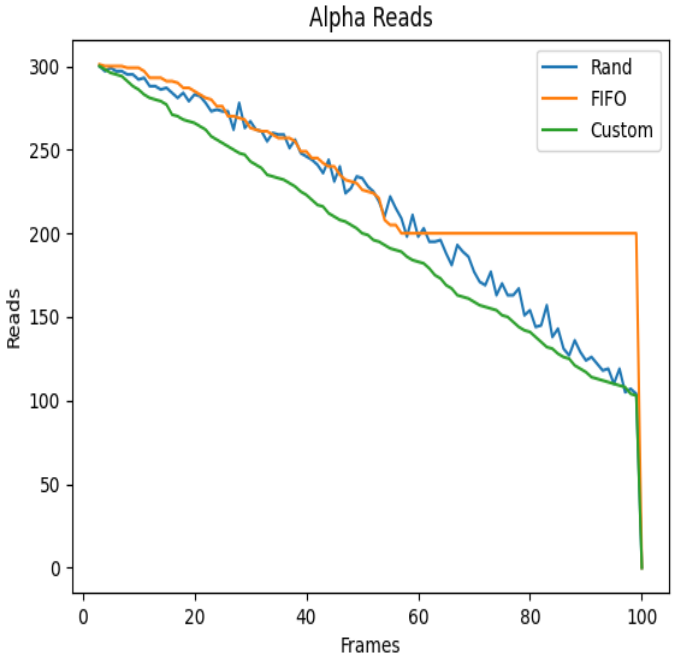
Once the page table is full, alternate between replacing the page at the (nframes)th index and the (nframes-1)th index in the page table. For example, if there are 50 frames, once the table is full, the algorithm alternates between replacing the 49th and 50th frame when it needs to, keeping 1-48 permanently intact. To accomplish this, you must just keep a running “alternator” number that alternates between 0 and 1 every time a page is replaced, and then replace the page at index [nframes - alternator - 1] (-1 is for 0-based indexing).

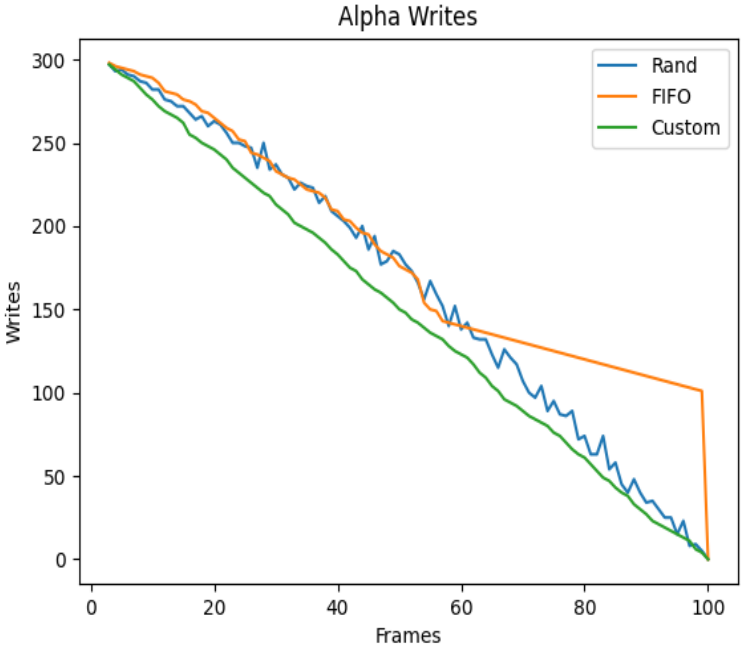
Section 3: Summary of Findings (graphs on following pages)

*When counting page faults, our group decided to include page faults that resulted because of incorrect permission bits.

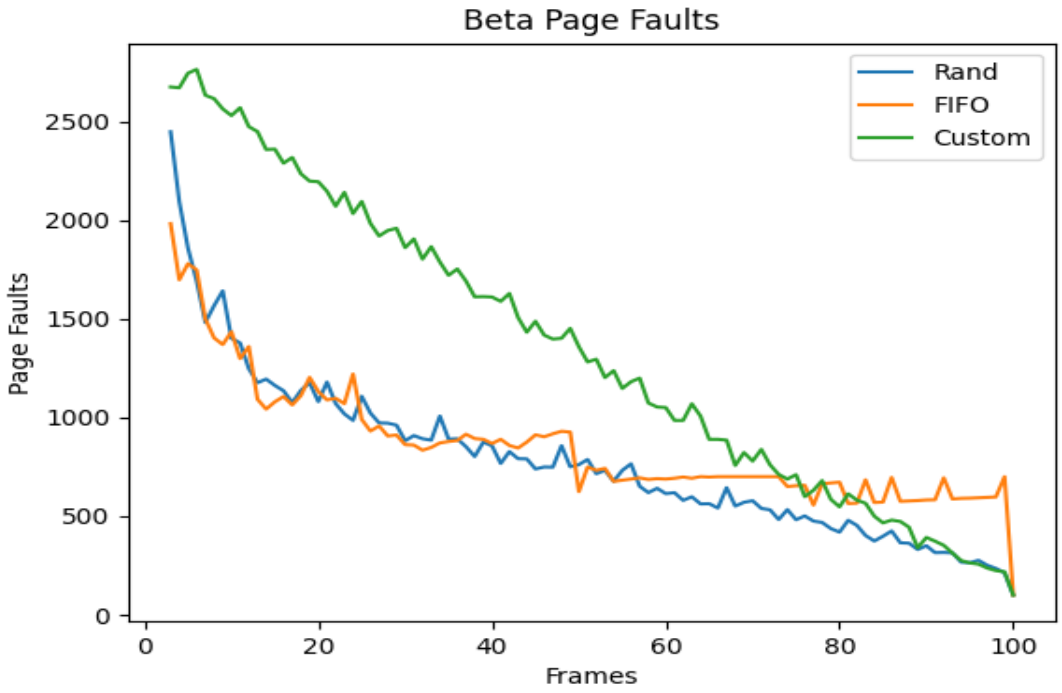
Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton

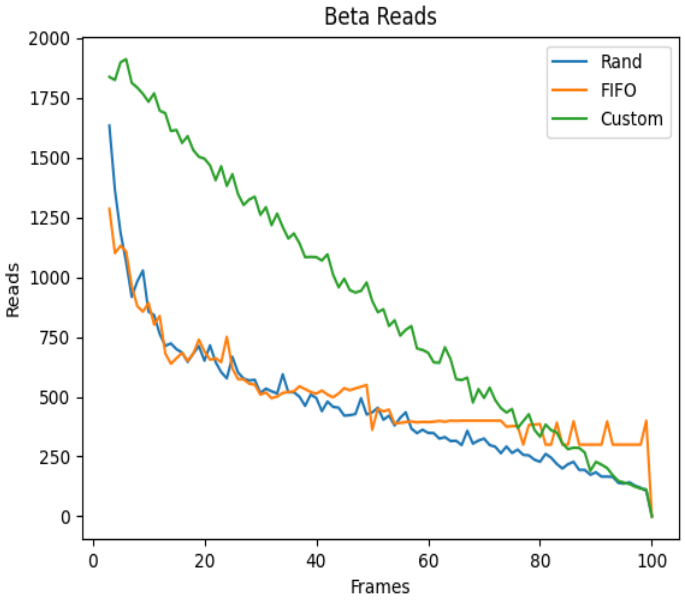


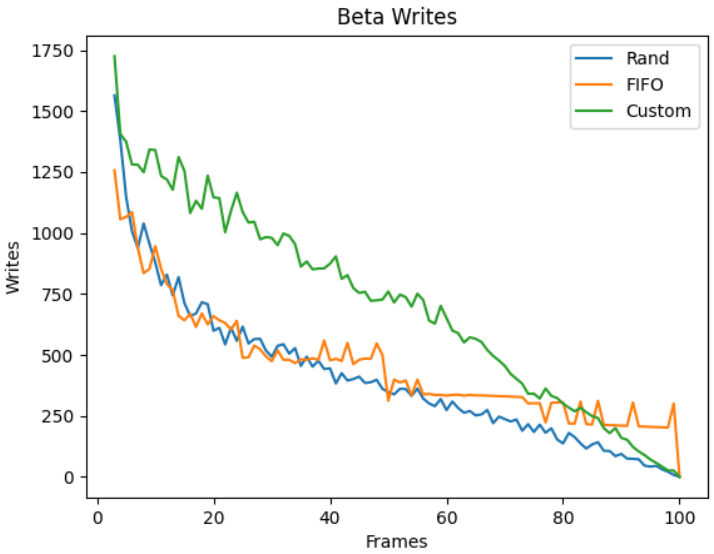




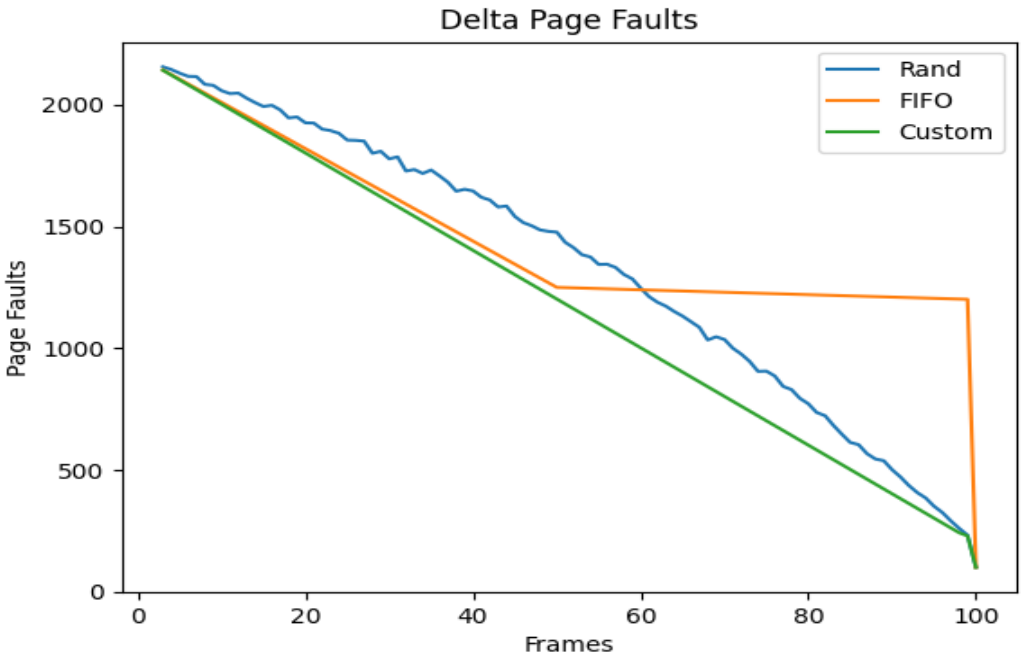
Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton

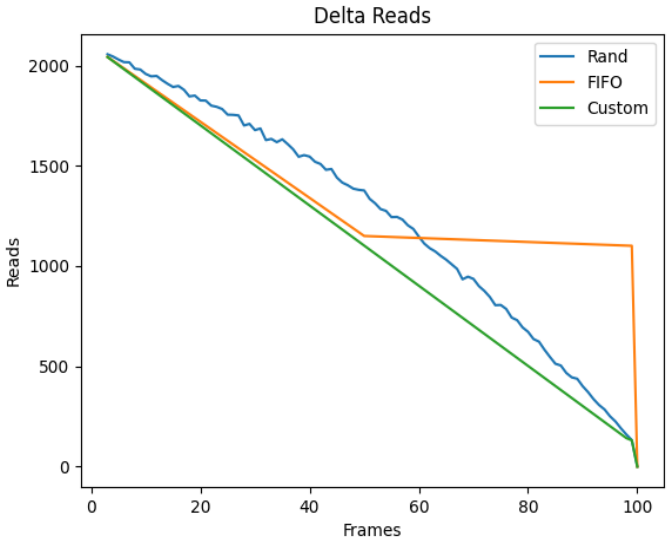


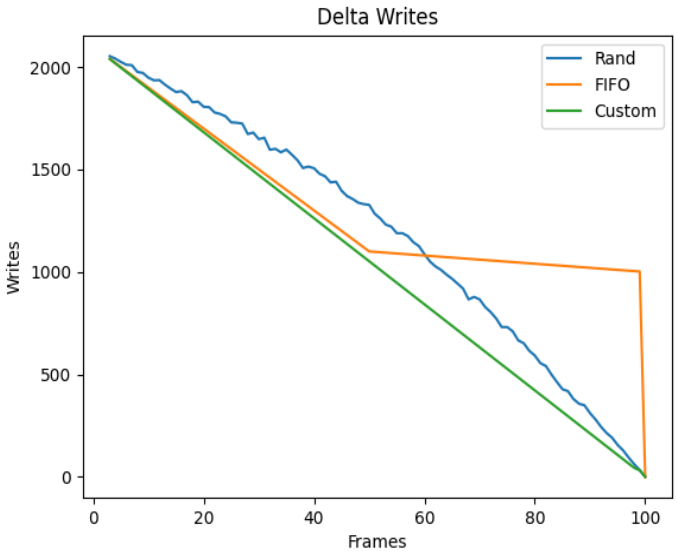




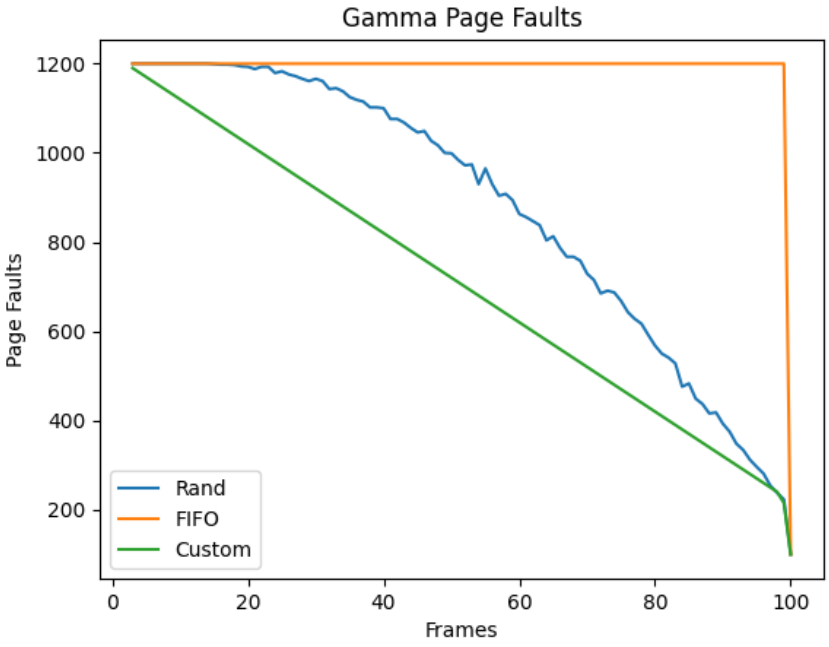
Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton

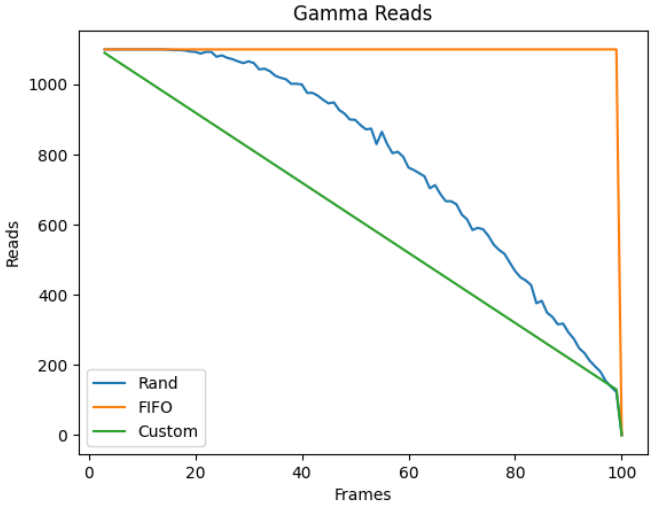


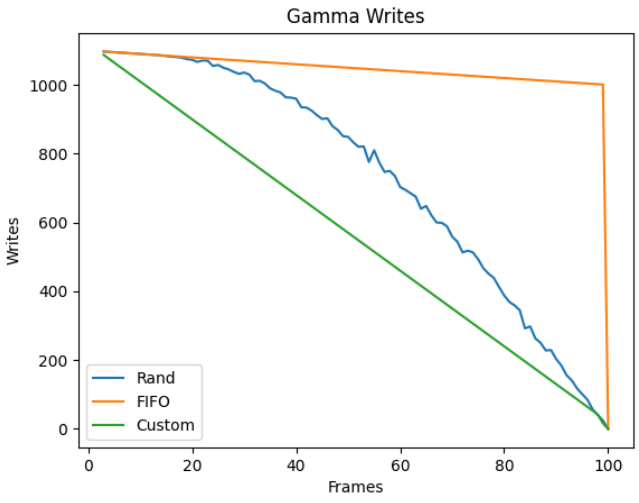




Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton







Operating Systems
Project 4
Group Members: Dylan Breslaw, Johnny Carr, Carter Goldman, Nick Newton

Section 4: Conclusion

Each replacement policy (rand, FIFO, custom) had a unique curve for each program that it was run on (Alpha, Beta, Delta, and Gamma), however, the shape of the graphs representing the page faults, disk reads, and disk writes were very similar for each policy for a respective program. As a result of this, we will talk about how each policy performed for each program holistically in terms of efficiency.

When testing our different algorithms on the **Alpha** program, our custom algorithm outperformed the other two on all metrics for virtually all inputs. Rand and FIFO criss-crossed until about 60 frames were used, at which point FIFO plateaued on all metrics because the random indexing used by the Alpha program made it less likely that any page present in the page table would get multiple hits.

The **Beta** program is the only one in which the other algorithms outperformed the custom algorithm, although as it approached 100 frames the custom algorithm became more competitive and eventually outperformed FIFO. Rand worked well for Beta because the quicksort algorithm that was employed made it so that the random fill of the page table ended up hitting quite often.

For the **Delta** program, both the custom and FIFO algorithms were very efficient for under 50 frames, however, once over half the page table was filled, FIFO flatlined because there were no more hits on the page table. The custom algorithm continued to perform very efficiently, while the rand algorithm lagged behind both but eventually outperformed the FIFO algorithm after about 60 frames. Custom performed very well throughout because the nature of the Delta program combined with the buffer in the custom policy allowed there to be many hits in the page table.

The custom policy had very similar results for the **Gamma** program to the Delta program, and the buffer allowed it to be very efficient again. FIFO performed horribly for the Gamma program, because it had a similar effect to what happened at the 50-frame mark for the Delta program, however it started immediately. The rand policy performed consistently worse than the custom policy, yet consistently better than the rand policy for the Gamma program. This is because the random aspect allowed it to get hits a relatively average number of times, whereas FIFO rarely if at all had any hits and the custom algorithm had hits most of the time.