Homework #2

Illya Starikov

Due Date: April 6th, 2017

For this assignment, the tasks were to simulate a memory manager with three page replacement algorithms

1. Clock Page Replacement
2. Least Recently Used Page Replacement
3. First In First Out Page Replacement

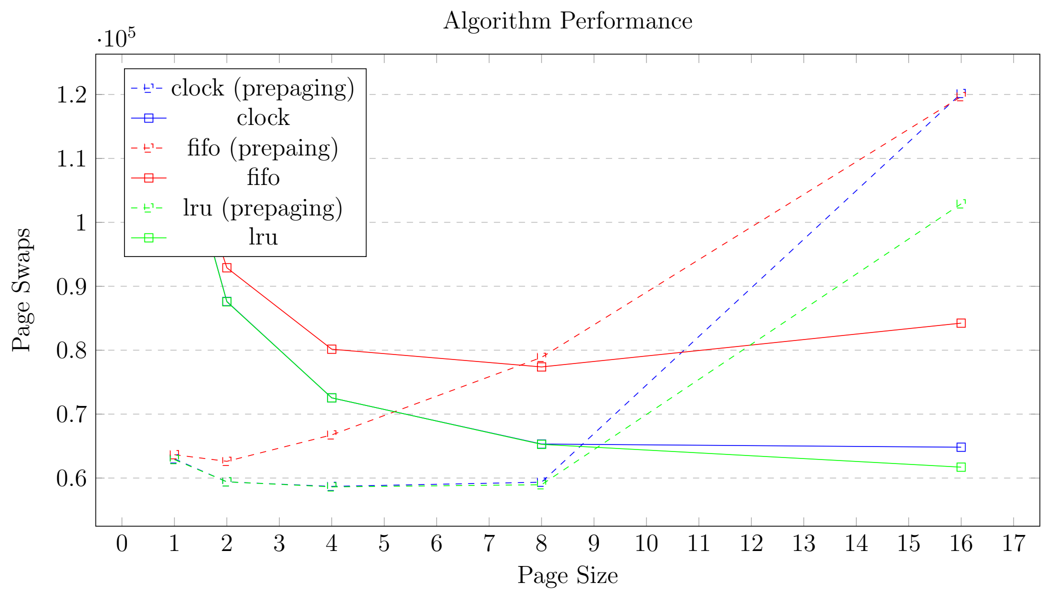

Figure 1 – Page Size vs. Page Swaps

This was accomplished by creating objects for the individual pages (eloquently named `Page`), programs (`Program`), and the memory simulator. The individual algorithms were implemented as methods of the `MemorySimulation` class.

The results can be summarized by Figure 1. For the most part, the page swaps ranged from 60 000 to 120 000. The different page sizing affected the algorithms the same, *depending on if there was prepaging*. For prepaging, higher page size increased the page swaps. The inverse also holds, for on demand paging, the algorithms faired better for higher page size. Overall, *Least Recently Used* and *Clock Based Paging* with prepaging worked the best for page sizes of 2, 4, and 8. The worst was *Clock Based Paging* and *First In First Out* with prepaging. The median solution was *Least Recently Used* with prepaging.

In retrospect, one algorithm was no significantly more difficult to implement than another. Conceptually, the *Least Recently Used* algorithm was the most difficult, and even that one was not hard. *First In First Out* was the easiest to implement, but was the worst performance wise. *Clock Based Paging* was the hardest, but had some of the best performance.

Overall, the complexity of the algorithms wasn't the deciding factor on which one to use in day-to-day situations. With very high page swapping happening in an operating system, the deciding factor is the performance.