Robert Higgins
CS 3800
Assignment 2

The goal of this exercise is to understand and apply three common paging algorithms by simulating requests to an operating system for memory under its control. Though the paging algorithms examined in this paper are all similar in nature and complexity, they exhibit trends which can be examined to find the behavior, advantages, and usefulness of the three different algorithms implemented, First in First out, Least Recently Used, and Clock. The details of how these algorithms were implemented are best understood by looking at the source code itself and only the results will be reviewed for now.

Demand Paging

Page Size	FIFO	LRU	Clock
1	117879	116886	117042
2	92800	87517	87574
4	80613	72820	72836
8	77910	65411	65412
16	84691	61790	65373

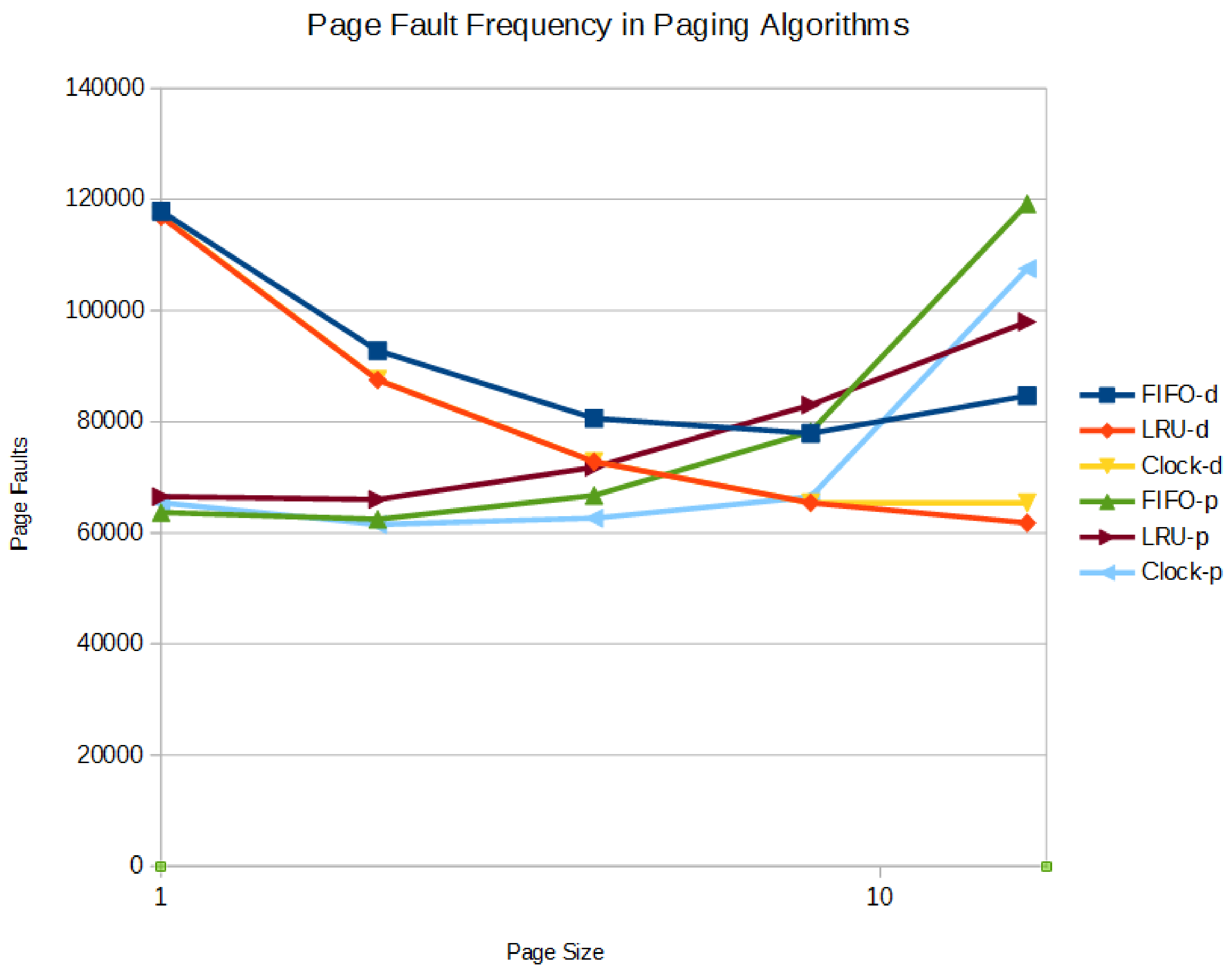
Prepaging

Page Size	FIFO	LRU	Clock
1	63684	66526	65446
2	62452	66006	61524
4	66745	71815	62638
8	78213	83034	66480
16	119223	97986	107552

As is seen in the tables above, each algorithm was tested for various page sizes and with optional prepaging in order to test their efficiency in a variety of conditions. This data represented as a graph allows for easier understanding.

By observing the graph below, several things are immediately obvious. First of all, when looking at the page sizes in relation to the use of prepaging, it is apparent prepaging works best with smaller pages sizes, while demand paging works better the larger the page gets barring processing overhead. This behavior is easily explained by with the probabilities. Smaller pages are less likely to contain other data that may be useful later, so many are eventually paged out from disuse. Larger pages with prepaging have the inverse problem, as each page contains so much data that might be used later, every time it is paged out, which if often due to the limited number of pages that can fit in memory, it must be retrieved again for other data contained in the page. Essentially, prepaging with large pages removes a large amount of relavant data from main memory when it still may be accessed in a relavant time period.

Apart from this, it can be seen that, in general, the more complex the selection algorithm, the more effective it is. It is important this occurs, as a task as common as memory allocation needs a low overhead to effectiveness ratio to do its job well. Overall, the clock method is probably the best in this test case. This is most likely due to the 'second chance' mechanism contained in the algorithm and the repeated accesses after delays present in the test trace.



Finally, it is very important to note these results are only representative of the data provided. The access sequences provided, for the most part, take advantage of locality very well, so prepaging and larger page sizes are rewarded to some extent. Were the access order to be, for example, completely random, prepaging would have not been nearly as useful and larger pages would have been even more inefficient given the likelihood of accesses being outside of their borders. The most important thing to note is the application of locality, both to this theoretical random set and real use. In a random access case, locality proves mostly useless, but in actual applications, sequential memory access is extremely common, so techniques like prepaging are rewarded greatly in performance.