Louis Rozencwajg-Hays, Nicholas Gattuso, Kexian Wu

I pledge my honor that I have abided by the Stevens Honor System.

Analysis

What we expected:

Before we ran the tests, we expected LRU to be the algorithm that would result in the least swaps, followed by Clock, and then FIFO. LRU would be the most efficient because only the most rarely used pages are swapped out, leading to less page faults. FIFO would be the least efficient. Belady's Anomaly states that when using a FIFO page replacement policy, increasing the number of frames available can actually increase the number of page faults that occur. So, we believed when we have more pages, recently requested pages could remain at the bottom of the FIFO queue longer, which results in an increased number of page faults. Nevertheless, FIFO and Clock would likely have similar performance, because they have very similar implementations. We also predicted that the demand and pre-paging policies would produce very similar results. Additionally, we theorized that with a bigger page size, we would see more page faults. Because of proportional allocation, a larger page size will result in having fewer frames per process. Therefore, having fewer frames will result in greater number of page faults.

What we observed:

After we ran the tests, we found that while our deductions about the efficiencies of the algorithms were correct, our other conclusions were wrong. For demand paging, as the page size increases, the number of faults decreases until size 8. Afterwards, for FIFO and Clock, it begins to increase again. With pre-paging, LRU and Clock slowly decline until page size 8, but then suddenly spike when the page size is 16. FIFO, however, declines until 2, then slowly rises until 8, before spiking like the others. Additionally, demand starts at a high page fault rate, while pre-paging starts at a moderately low rate. Nevertheless, all algorithms start at almost the exact same value (demand and pre-paging have different starting values).

We observed that LRU is most efficient, then Clock, then FIFO, which is the same as what we expected. However, FIFO and Clock displayed a large increase in page replacements from page size 8 to page size 16 in both cases. Also, with pre-paging, we observed that LRU starts increasing for page size greater than 8. We believe the reason behind this is that there are not enough frames available to satisfy its minimum working set. In other words, not having enough space for a working set will cause thrashing. Since the number of frames is equal to the size of the memory divided by the page size, increasing the page size will proportionately decrease the number of frames. In our program, we will hold 512 memory locations. If the page size is 16, we only hold 3 pages for each process, which results in busy swapping pages due to a lack of space.

Another discrepancy between what we predicted and what was observed is that demand paging starts at a high page fault rate, while pre-paging starts at a moderately low rate. We think this might be because pre-paging attempts to predict the pages that will be needed in the near future, and page them in before they are actually requested, which will increase the chance that the memory location we try to access is held in the main memory.

Time Complexity:

LRU was the easiest algorithm to implement. We updated the last-accessed time each time we accessed a page. When there was a need to swap pages, we swapped the oldest page with a new page. We simply did a linear search to find the page with the oldest timestamp. Therefore, the time complexity for this approach is O(n).

FIFO and Clock had similar complexity since they both used the same data structure (Linked List). Clock also had the extra overhead of keeping track of the reference bit for each page, the hands of the clock moving around in the linked list. For FIFO, when there was a need to swap pages, we popped off the page that was loaded first (the first element of this linked list) and pushed a page to the end. For Clock, when there was a need to swap pages, if the page at the beginning of the linked list did not have its reference bit set, then that page would be popped off, a new page would be added to the end of the list, and the pointer would point to new head node. The time complexity for FIFO and Clock are both O(n) in our implementation.

Randomness:

If a random memory access trace had been supplied, it is possible that the results would change by a considerable margin. This is because by having random input, the program may be lucky or unlucky with regards to the order in which the pages are inserted, leading to noticeably more or less page faults. Overall, in the worst-case scenario, we believe the performance of the three algorithms will be worse. Nevertheless, LRU will likely have a page fault rate similar to what was observed in this experiment, due to its greater adaptiveness to randomness. FIFO and Clock, however, will show noticeable changes depending on how the pages are inserted. Clock, especially, will have more page faults than FIFO. Additionally, with pre-paging, the number of swaps will increase dramatically, because pre-paging’s predictive nature will have less of an effect on a random trace.