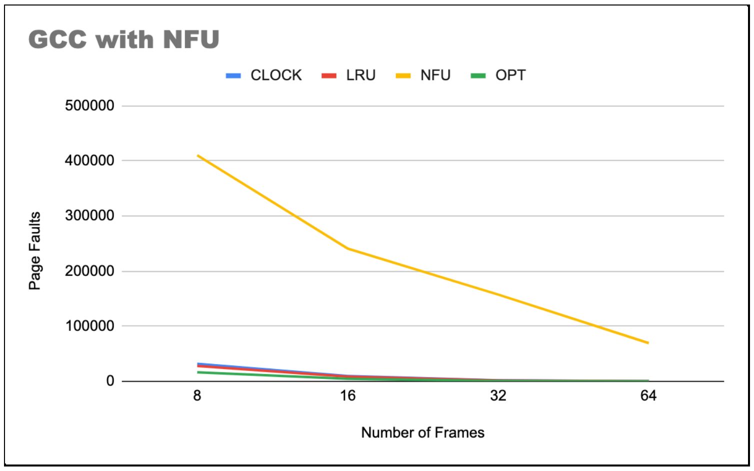
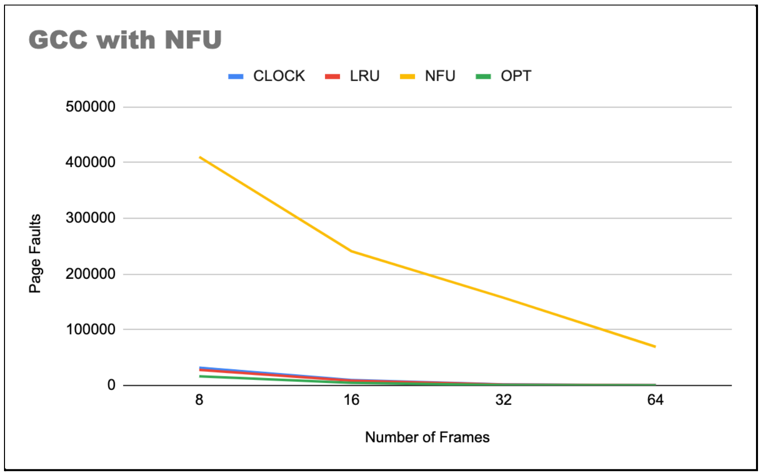
Given the provided graphs running GCC.trace file which contained 1.7 million instructions, we can observe that Clock and Least Recently Used (LRU) page eviction algorithms, compared to Optimal Page Replacement, would be best to implement in an operating system.



Both Clock and LRU consider the references a particular page has from the past. Clock operates much like Second Chance, but has a revolving counter that eliminates the overhead of moving data within some structure. LRU looks at the recency in which a page has been referenced in the past with the assumption that a page furthest away from being referenced has the likelihood it will not be used in the near future.

Both of these algorithms were reasonably simple to program and required minimal overhead. LRU outperformed Clock by a small margin in all frame sizes, so the choice to pick between the two would require consideration of other programming factors.



Not Frequently Used Page Replacement algorithm, as we can tell from the graph including this data, performed very poorly. This algorithm looked at the frequency the page was used in the past. The core concept is founded in the belief that a page that has been referenced more often in the past, the likelihood it will be used again. Given the data, we can clearly see this is flawed and allowed for much more page faults compared to the other algorithms.

Interestingly, we can improve this number by half contingent on the implementation of NFU. In the NFU implementation displayed in the graph, we reset the frequency value of the page when it is being evicted. In other implementations, like the one described in our textbook, NFU does not forget this value even after eviction. When we remember this value (not clearing it when evicted) the number of page faults drops from 400k to 200k.