Dane Halle
Project 3

Analysis of Page Faults using Page Replacement Algorithms

The results from performing each algorithm on the trace files with increasing number of frames are interesting. If we look at **Figure 1**, **Figure 2**, and **Figure 3**, we can see a general curve downwards as the number of frames increase. In **Figure 2**, we see that the LRU and Optimal are almost exactly the same in terms of output data for the four different frames. This makes sense that the faults would be lower because there are less times when an eviction have to happen the more frames are available. The only other times that faults occur is when you have to put a page into the table which can only every happen n_{unique} times where n_{unique} is the unique number of addresses within the trace file. The best algorithm found was to be Optimal but that is to be expected. The issue though is that is very costly as it goes over the entire trace in order to be so good. That means that the best, more practical algorithm that would be used by a real OS is LRU due to it having less (or at least very close) faults than Second Chance in every frame tried.

Analysis of Faults using Second Chance with 2-100 Frames

For my sanity, when plotting data resulting from this test, I only took when frames were 2, 4, 8, 16, 32, 64, 80, and 96 as that should give a good look at what the graph should look like in terms of shape. We have the results of the graph in **Figure 4**. Each trace starts off with a ton of faults and then decreases steadily. The only one that does not as significantly as the others was the gzip file. When I first got the output in a text file in a workable form to get the data needed, I was not sure how I would determine if there would be any point a Belady's Anomaly occurs. Then I figured I could parse though the file to determine if the page fault it was looking at was bigger than the previous and iterate over the entire dataset. This pointed me to frame 92 within the gcc.trace data. The faults stated was 458 and the one previous to it (frame 91) was 457. The frame 93 also had 458 faults before it went back to seemingly normal with frame 94 having 453 faults. No other traces appeared to have this anomaly.

Implementation of Optimal Algorithm

My implementation of the Optimal Algorithm is essentially the same as what the notes and slides states. I first parse over the entire trace to pre-process it in order to know which instructions use which pages and in which order. This has linear runtime of O(n). I then reset some elements and start the actual page replacement algorithm. The only time the pre-processing is called forward is at the beginning of every instruction when the instruction is removed as it was already used and when a page needs to be evicted. When a page needs to be evicted, it determines which page that is currently loaded will be needed the furthest in the future and evicts it. This takes $O(n_{frames})$ to run where n_{frames} is the number of frames available. This means the runtime for the algorithm is $O(n+n_{frames})$. This runtime does not account for the runtime of the general page replacement algorithm but rather just the changed needed to implement it with the optimal algorithm.

Graphs

For all graphs that look at Page Faults and Disk Writes as Frames Increase, the following legend is in place as I was unable to get them labeled on the actual graphs:

Blue line – LRU Algorithm
Red line – Optimal Algorithm
Yellow line – Second Chance Algorithm

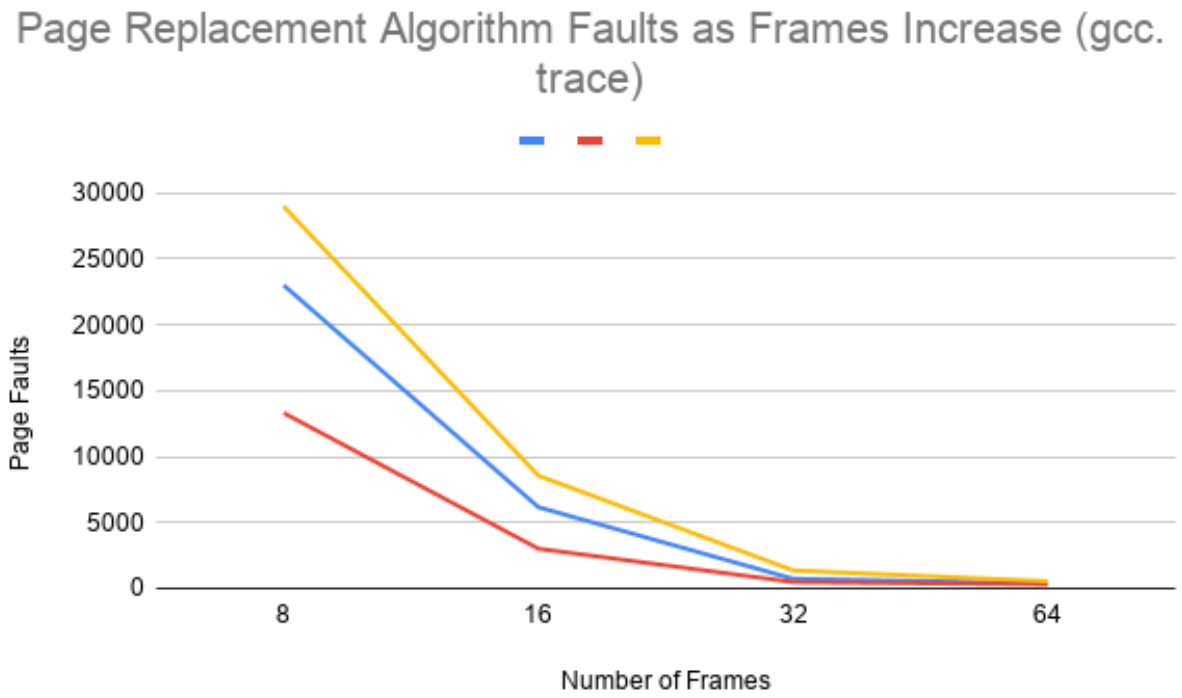

Figure 1: Number of Page Faults as Frames increase on gcc.trace

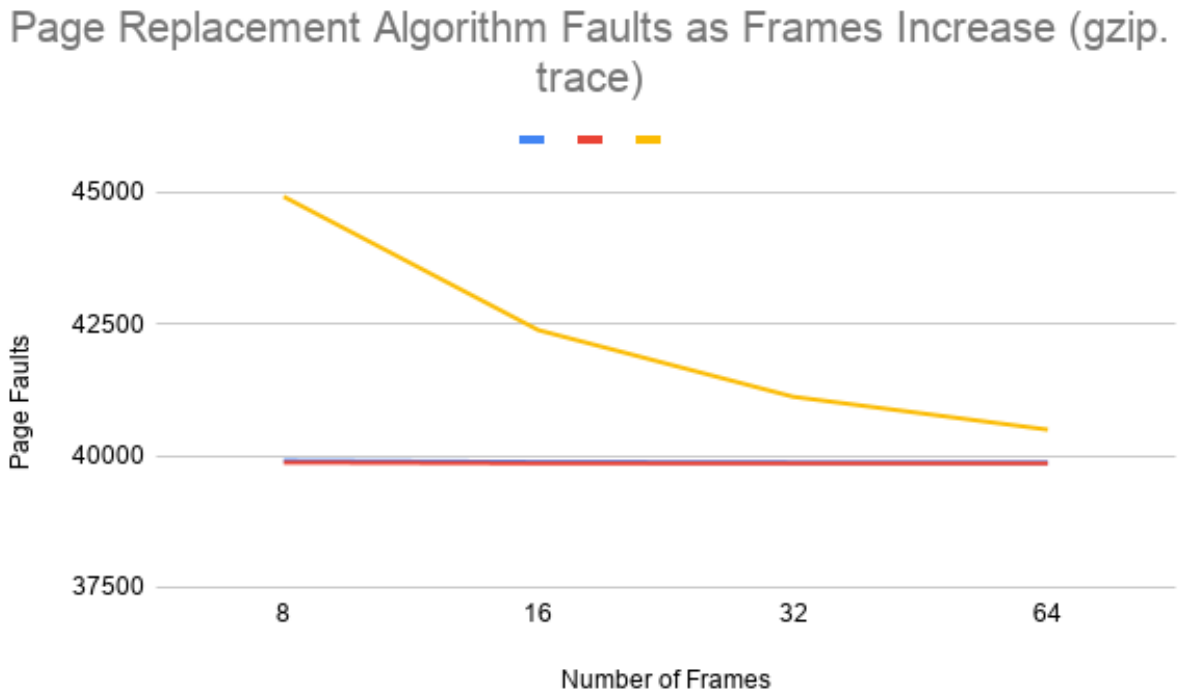

Figure 2: Number of Page Faults as Frames increase on gzip.trace

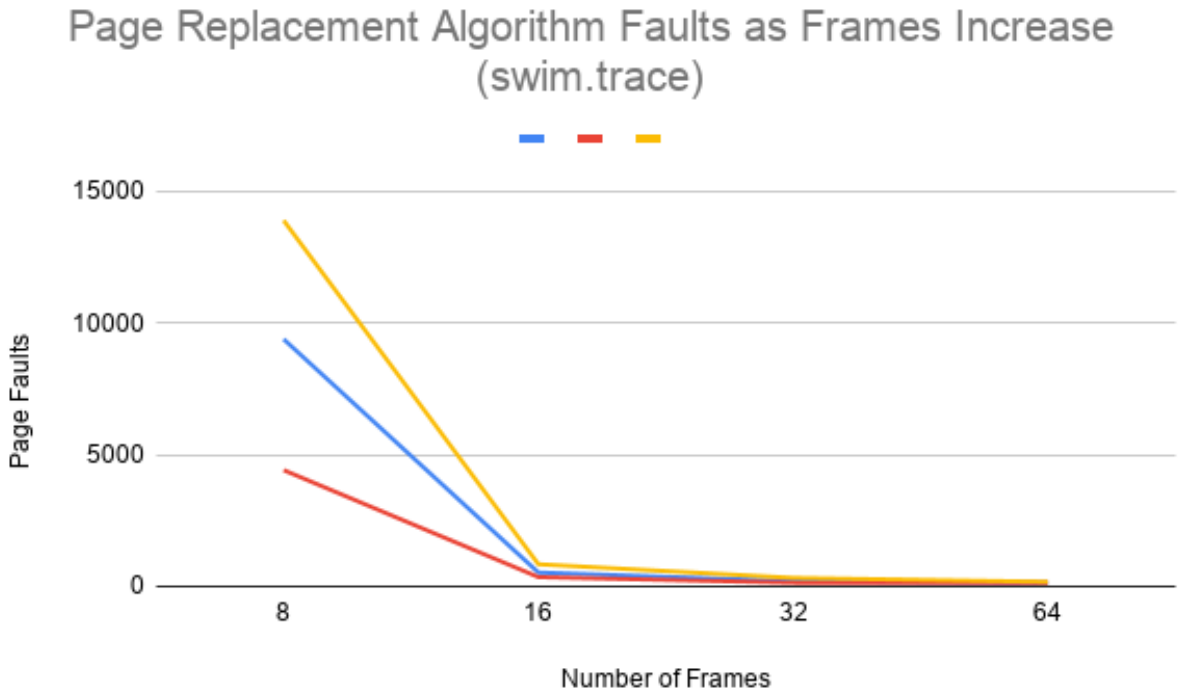

Figure 3: Number of Page Faults as Frames increase on swim.trace

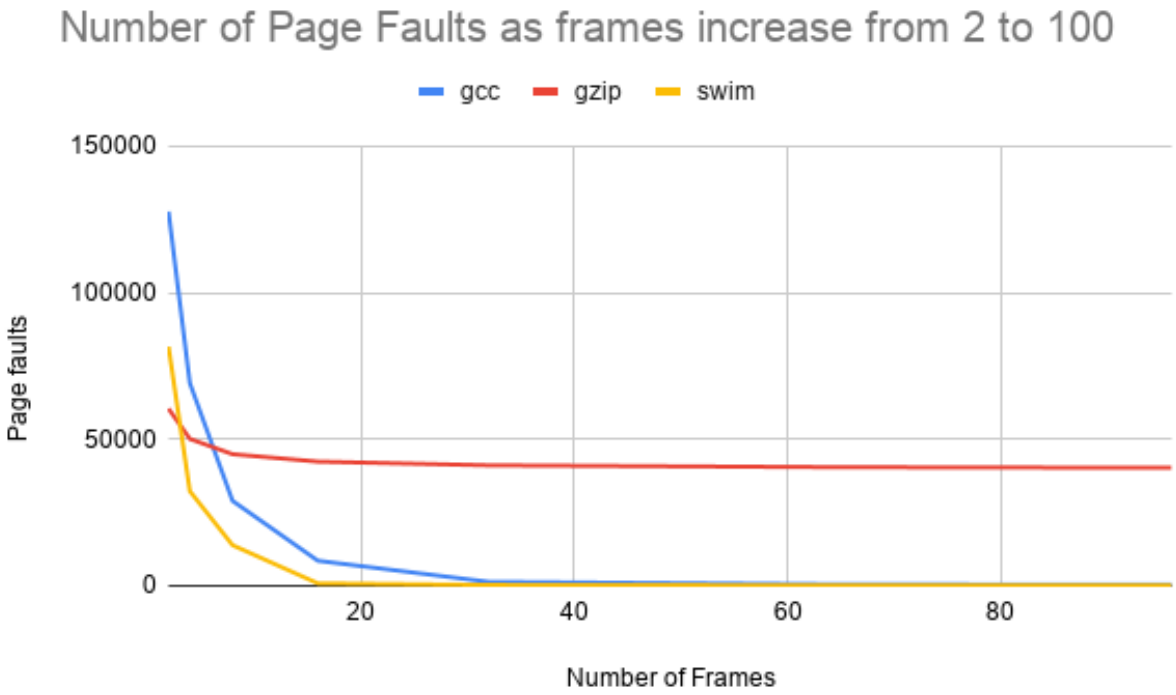

Figure 4: Number of Page Faults as the Frames increase from 2 to 100 using Second Chance Algorithm on all three trace files