Vinna Gu

Professor Khattab

CS1550: Introduction to Operating Systems

15 November 2020

Project 3: Second Chance Page Replacement

In this project, I implemented the Second Chance algorithm using only a linked list to store new pages based on the number of frames each time we encounter them in the .trace files that were given. Essentially, the algorithm follows something similar to a FIFO queue; therefore, using a linked list seemed the best data structure to use because rather than having to traverse through the list, we simply remove the oldest address that was added into the list which is the first item (the head). However, if the oldest address had a reference bit of 1, we would set it to 0, and set it at the end of the linked list to give it a second chance. Along with that, we had to consider if the dirty bit was set to 1 and if so, the number of disk writes will increase every time an address was removed.

With that said, to test different frames and page sizes, I ran several memory splits: 1:1, 1:3, 3:1, 3:5, 5:3. When the total of frames were equivalent to 1024 (based off of the project requirements), I included bar graphs to represent the 7:9 and 9:7 memory splits.

Based on the graphs created, it generally appears that the bigger the page size is, the fewer the page faults and disk writes. Along with that, the number of frames also contributes to this decreasing number of page faults. We can see that in the graphs, the number of page faults and disk writes significantly differ between 8 and 1024 frames, with 8 frames and page size of 4KB having up to 160,000 page faults, and 1024 frames and page size of 4KB having only up to 40,000 in comparison. This may be because multiple virtual addresses (excluding their offset) map to the same frame and given that the page size is so large, we do not have to do as many page replacements (thus no page faults), and only check to see if there was a disk write or not.

Looking at the bar graph with 8 frames and a 4MB page size, we can see that the page size and frames significantly affected how many page faults occurred. However, there is the case where the memory split with the ratio 1:3 has a higher number of page faults and disk writes than any other ratios. This is because one process (process 0) received two frames, whereas the other process (process 1) received six frames. Because of the uneven distribution of frames, this increase also applies to the 8 frames with a page size of 4MB. Although the frames is a major contribution of the number of page faults, the page size also plays into this and we can clearly see that between the two graphs that with a bigger page size, it also reduces the number of page faults. As a result of this, having a somewhat even distribution of frames with a large page size can significantly reduce the occurrences of these faults; therefore, the memory split 5:3 would be highly preferred with 8 frames and a page size of 4KB each. For 8 frames and page size of 4MB, the ratio 3:1 would be the preferred choice. Between the two; however, it would be best to go with the 8 frames and 4KB page size for better performance unless performance is not an issue.

With 1024 frames with a page size of 4KB as well as the page size of 4MB, it strangely appears that the number of page faults were consistent no matter what the memory split was. I’ve tested this several times with various trace files given just to see if it was an issue on my end, and the results were the same. However, in comparison to the 8 frames with 4KB page size, I would say that the low page fault rate is better, especially with 1024 frames and 4MB page size. In addition to that, we’re able to add as many addresses as we would like since there’s a bigger chance of them being in the same frame. Even though the low page fault rate is better, there is a con to having a big page size. We run the risk of having a lot of unused space due to the large page size, thus the possibility of having addresses from being added to memory non-contiguously.

As a result, it ultimately depends on the users on what they want since there are some pros and cons of having a big number of page frames and a big page size. To iterate what memory split works best for each frame and page size, an 8 frame with 4KB page size would be best with a memory split of 5:3. For 8 frames with 4MB, the best would be a memory split of 1:3. For 1024 frames with 4KB or 4MB, you can not really go wrong with any of them. They have very slight differences especially with 1024 frames and 4KB where the highest page faults totaled up to 40,178 while the rest were 40,168. 



