Allison Raines, Kathryn Lutz, Gabe King, Eric Layne
Team: kage
April 19, 2018

Purpose

These experiments were conducted to test the efficiency of page replacement algorithms, specifically FIFO, random replacement, and an algorithm that we designed (custom). When determining efficiency, page faults, disk reads, and disk writes were used as a metric to compare the effectiveness of the algorithms used.

Experimental setup

These experiments were run on the student02 machine. The command line arguments we used were:

```
./virtmem 100 X <FIFO|rand|custom> <scan|focus|sort>,
```

where X is the number of frames used.

Custom page replacement algorithm

Our custom algorithm utilizes a vector that keeps track of how many evictions occur on a given page. Each time the page fault handler is called, the value of the page fault occurrences increases by 1 on that specific page. When it comes time to evict a page, the algorithm searches through the vector for the page with the least evictions so far. Ties default to the last count of that value in the vector, and if all values of page fault occurrences are 0, we select one at random (which only occurs on the first eviction). Then, the page with the least number of page faults thus far is evicted.

The motivation behind this strategy is to avoid the problem that is often encountered with FIFO and random eviction methods: kicking out an important page that is likely to be referenced again. In keeping track of the number of times a page enters the page fault handler, we make a predictive guess as to which pages are least likely to fault again and evict these chosen pages. This algorithm, of course, is not perfect as it is nearly impossible to predict which pages will fault in the future. However, the point is that the algorithm attempts to be predictive in comparison to the FIFO and rand algorithms.

It is important to note that this custom algorithm is concerned with frequency (if a page has been accessed many times) rather than recency.

Page Faults

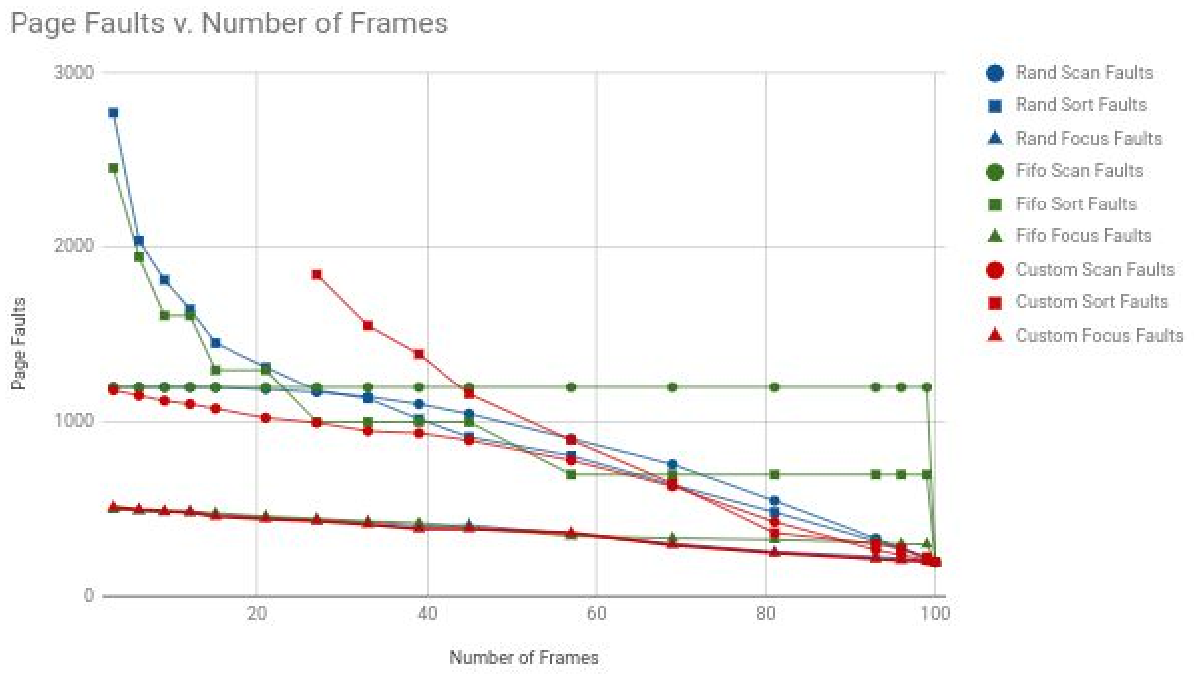

Figure 1. Page faults v. number of frames.

In our paradigm, a page fault occurs for one of two reasons. The first is representative of an actual attempted memory access where the desired data is not in virtual memory, whether due to it never having been loaded in the first place or it having been evicted to make space for other data. The second, auxiliary function of our page fault handler is to ensure modified pages are properly written back to physical memory. To achieve this, our handler is also triggered when an attempt is made to write to virtual memory which has only the READ flag set. In this case, there is no actual page fault occurring. Instead, our handler is just the mechanism to flag the data in virtual memory with the WRITE bit to ensure physical memory remains faithful after its eviction. When virtual memory is full and more pages cannot be added, a page needs to be evicted from the frame table in order to make room for the new page.

The FIFO strategy for page eviction takes the first page that was in the frame table out and replaces it with the new page. The random strategy for page eviction randomly chooses which page to evict from the frame table. The custom strategy decides which page to evict by taking into account all the times that the page had been evicted in the past.

In terms of page faults, FIFO performs considerably worse than random replacement, as seen in Figure 1, especially when the scan and sort programs are run. The FIFO algorithm utilizes a queue where the first page that was placed in the queue is the first one evicted. Clearly, as indicated by our data displayed in Figure 1, the FIFO algorithm tended to evict pages that were referenced more frequently than those pages that were evicted using the random method.

Regardless of the eviction strategy used, the most optimal strategy involves the predictions of which pages will be accessed in the future (this is impossible, but the custom algorithm attempts to do this given a history of faults for a given page) and evicting the page that will no longer be used or be used the least amount of times. This requires knowledge of how the program's memory will fare in the future but in our case, the next best strategy is choosing a page that has the fewest number of page faults. This strategy was optimal when scan and focus were used because the program attempted to predict which frame was the least likely to be accessed again given its history.

Scan

For the scan program, page faults as a function of frame count is a linear function for each of the algorithms. The custom algorithm had the best performance, followed by random eviction and FIFO. Because the scan program accesses memory sequentially, FIFO results in the flat line of faults because it replaces frames in a similar manner. The program scans through virtual memory, so the page replacement algorithm must continually overwrite the first frame added to the frame table, resulting in a constant string of faults no matter the frame count. Rand performs slightly better, as it inserts the data into any frame. Accessing the next virtual memory through scan might point to a previously assigned frame which was not overwritten by the page fault handler. As the number of frames increases, the likelihood that the frame is overwritten decreases, resulting in decreased faults. Finally, because the custom algorithm keeps track of the number of evictions on each page, it minimizes faults by overwriting the frame that has had the least number of evictions and therefore the least amount of faults.

Sort

The reason the sort algorithm performs the worst for the FIFO eviction strategy is because the pages are being accessed from memory in sorted order. The data leads to the conclusion that FIFO performs relatively poorly when the page table is sorted and the number of pages exceeds the number of frames by a considerable amount. Since the page table is sorted repeatedly, the order of the pages that the program is attempting to access is being changed. The order of the virtual memory accesses is essential to FIFO's performance, so by changing the order repeatedly, the order of disk accesses is greatly altered, and the number of page faults could increase from this. For random replacement, the order of the accesses does not necessarily matter as much, given that the pages in the frame table are evicted at random. Random eviction, again, is simply based on luck, and the order the pages are being placed into memory is not as important as it would be for FIFO.

Focus

When a page fault occurs, disk must be accessed. As is further evidenced in the disk reads and disk writes graphs (Figures 2 & 3), the relationship between the number of page faults and the number of frames is more influenced by the program (focus) than the corresponding eviction strategy. This makes sense because the focus program chooses pages in such a way that page faults occur less frequently to begin with and therefore do not depend on an algorithm as much as the other two programs to evict those frames. In summary, fewer page faults means that there is less emphasis on an eviction strategy to resolve those page faults.

Disk Accesses

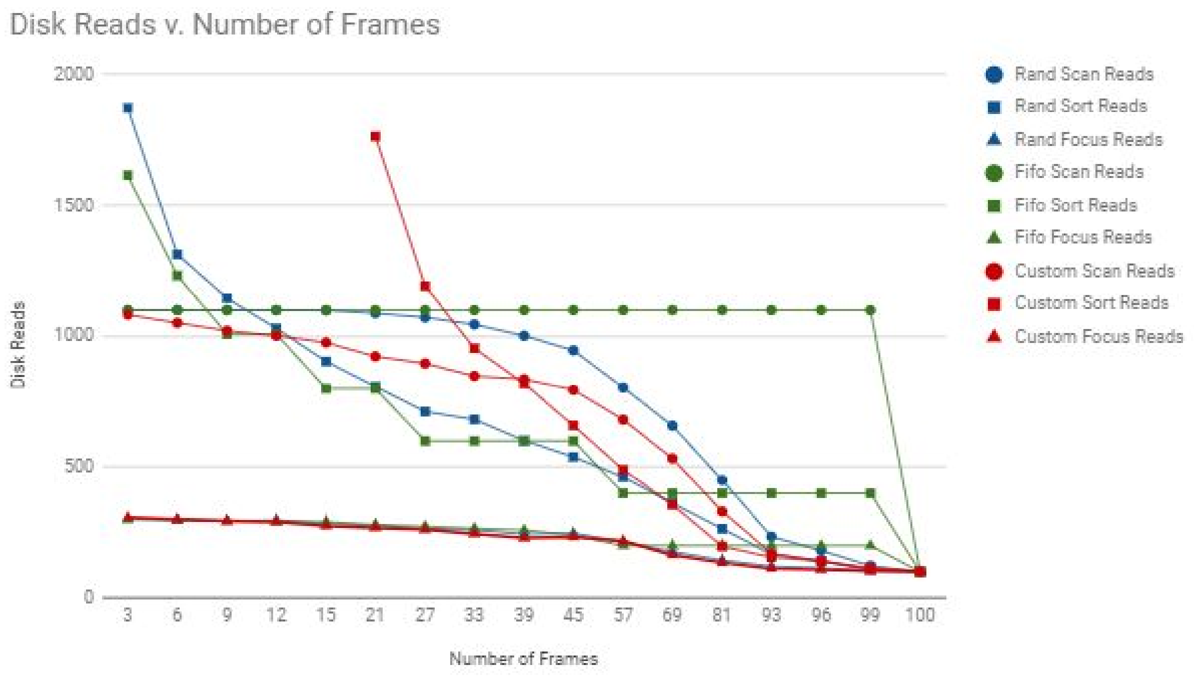

Figure 2. Disk reads v. Frames used.

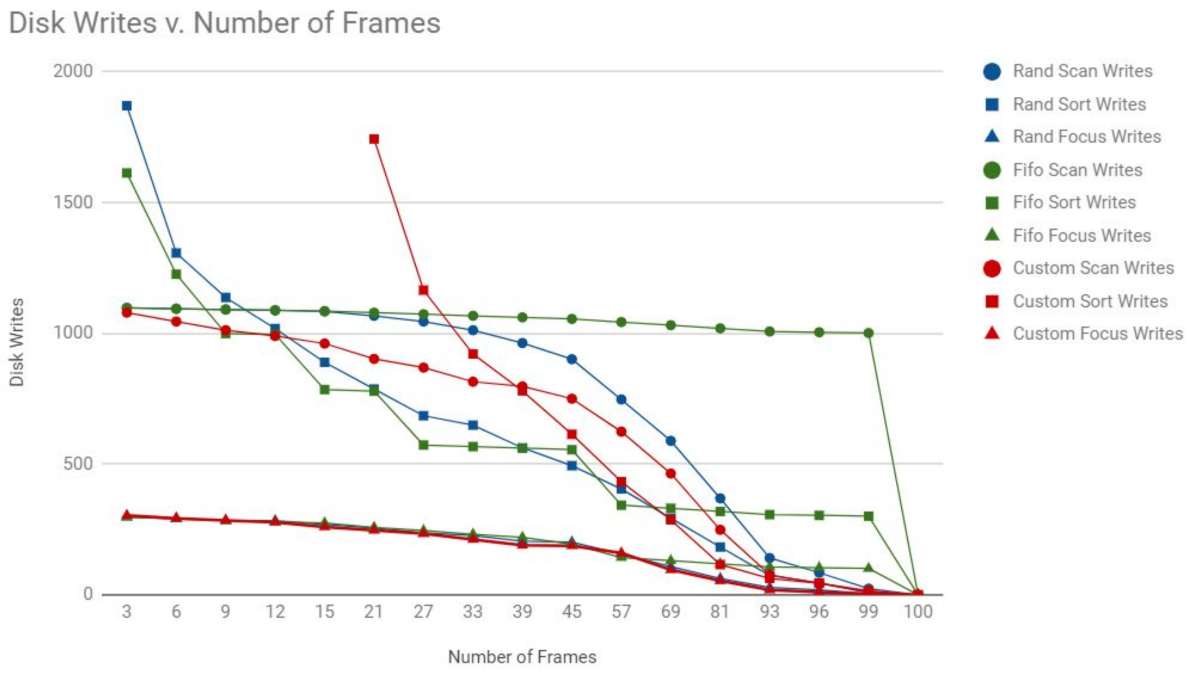

Figure 3. Disk writes v. Frames used

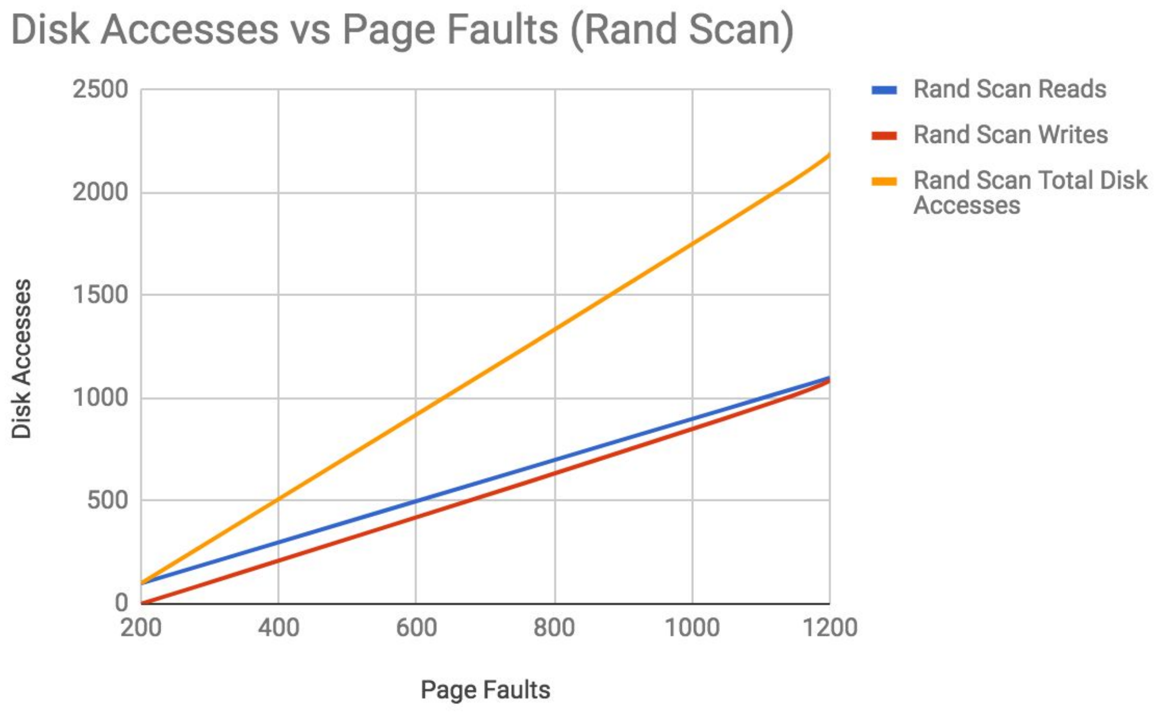

Figure 4. Number of disk accesses v. number of page faults, when the eviction strategy is random scan.

When a page fault occurs, physical memory (disk) needs to be read from/written to in order to allow a page to be placed into virtual memory, and the number of page faults and the number of disk accesses are directly related. This relationship can be seen in Figure 4, where the number of disk accesses and page faults for the rand scan eviction strategy can be seen. This trend is followed when other eviction strategies are utilized.

This close relationship is due to the inherent necessity for data from the disk when a virtual memory miss (page fault) occurs. In the case of an empty page, a compulsory miss, a disk read into virtual memory will always accompany the fault. In the case of all frames being full, triggering a conflict miss, there will always also be a disk read to populate the requested data in the page table, and sometimes an additional disk write (if the W bit is set).

Given the above discussion regarding page faults and the three eviction strategies, it is expected that the trends on the graph of disk accesses v. pages used (Figure 3) follows similar trends as the graph of page faults v. eviction strategies.

<u>Scan</u>

In the scan program, the disk is read sequentially. Therefore, the numbers associated with disk accesses follow a similar pattern as the number of page faults. We have concluded that the number of disk accesses, which happens only when a page fault occurs, can be attributed to the same reasons why the page faults happen -- because the pages are being loaded into virtual memory in sequential order.

<u>Sort</u>

The numbers associated with disk accesses follow a similar pattern as the number of page faults. The data leads to the conclusion that the number of disk accesses, which happens only when a page fault occurs, can be attributed to the same reasons why the page faults happen -- this is because the pages are being loaded into virtual memory in sorted order.

<u>Focus</u>

As is shown in Figure 2 and Figure 3, the focus results for each of the algorithms is nearly the same (there is very little difference in the number of disk writes and disk reads for each of the algorithms involved with focus). This indicates that this program is hardly influenced by the algorithm used; because of this, we can draw the conclusion that in using the focus program, the number of disk reads and disk writes is influenced by the program rather than the corresponding eviction strategy used.