Sam Rack
CSE 30341 Operating Systems
Project 4 Report

Experiment Purpose

The experiments were run for all three page replacement algorithms on the three programs with a variety of frames available in the memory. Running these experiments and recording the number of page faults, disk reads, and disk writes allows for a comparison of the three page replacement algorithms on benchmark programs. Trends can be observed for the three algorithms, and the differences in performance can be analyzed for programs that access memory differently.

Experimental Setup

To run these experiments, I wrote a bash script that ran each of the programs on each of the algorithms for 12 different frame counts. The output of the script was saved as a csv file so it could be graphed. The script, parse.sh, is included in the project4 directory.

The experiments were run on the student machines. The command line arguments were the same as were specified in the project document, of the form: *./virtmem <numPages> <numFrames> <rand, fifo, custom> <focus, scan, sort>*.

Custom Page Replacement

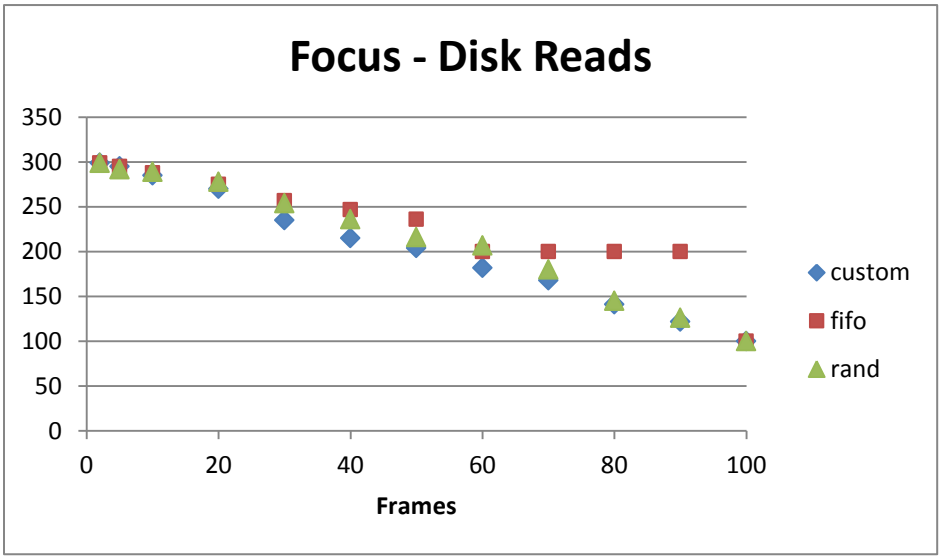
The custom page replacement algorithm used was similar to the FIFO algorithm, but it included some consideration for if a page would have to be written back. Additionally, the FIFO order was actually a least recent page table update (different because it was moved to the end of the list when the write bit was set). The algorithm works as follows:

(1) at start of program, initialize an array of size <numFrames>, arr, to 0
(2) when a page fault happens, decrement all elements in arr
(3) when a page fault happens (or write bit is added), reset arr[<frameBeingChanged>] to 0
(4) when a victim frame is selected:
    (a) find minimum values in arr for frames that have write bit set and frames that only have the read bit set
    (b) select one of these two minimum frames – if the no write frame is only greater by the write frame by a threshold (for my implementation, half the number of frames) or less, then select the no write frame; otherwise select the frame that has to be written

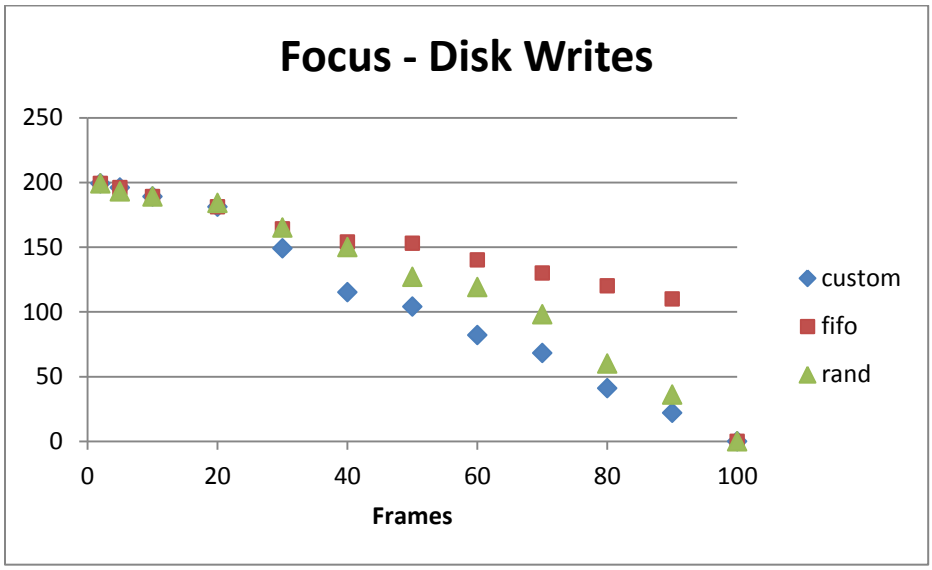
This algorithm will in general result in a lower amount of disk writes because priority for selecting a victim frame is given to the frames that don't have to be written back to disk. However, if the only frames that do not have to be written back were recently read from disk, a frame that must be written back is selected.

Note: The number of disk reads is equal to the number of page faults, since changing the write bit is not actually a page fault (definition: a page fault occurs when a page is not found in memory and must be loaded from disk first). Therefore, I only provide graphs of Disk Reads and Disk Writes.

Focus Program Results

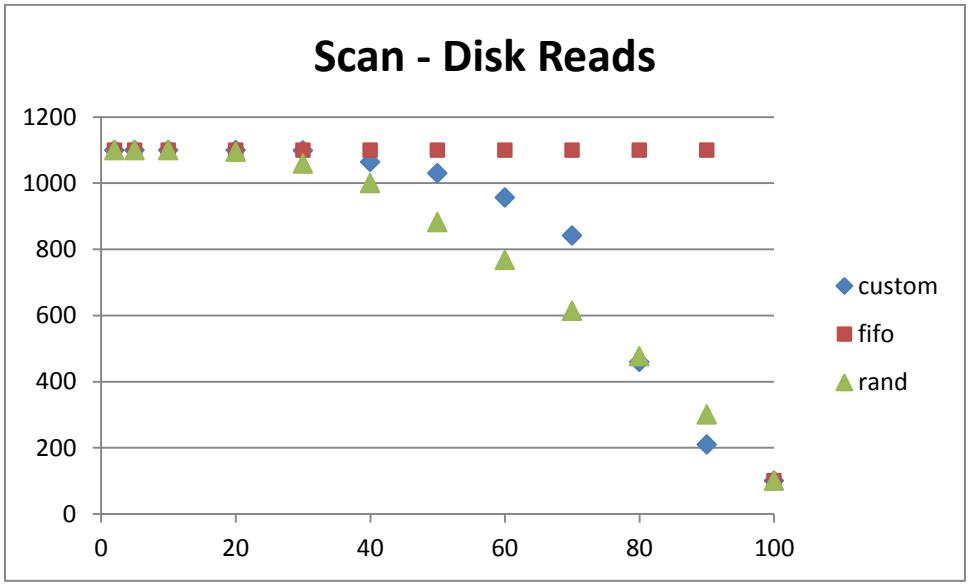


The custom algorithm performs slightly better on number of disk reads/page faults than the other two algorithms for some middle-range number of frames. The program works in a locality, so FIFO is ineffective (will get rid of a page in memory when it is likely to be used soon again). The custom algorithm works better than FIFO likely because of the update in its use on the change in the write bit.

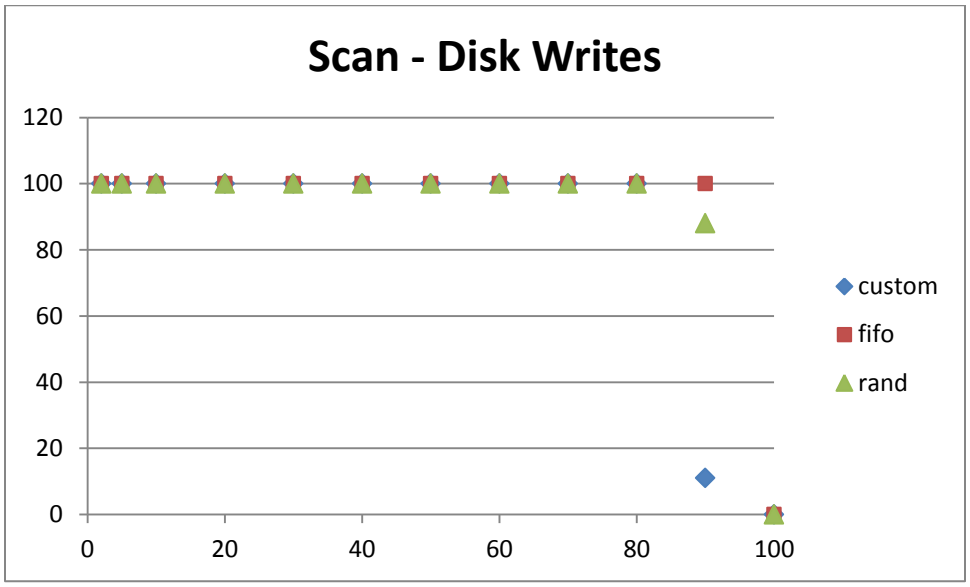


The custom algorithm performs better on disk writes than FIFO and random in most cases. This is a result of the custom algorithm taking into account whether or not a frame will have to be written to disk. The other two algorithms do not take this into account.

Scan Program Results

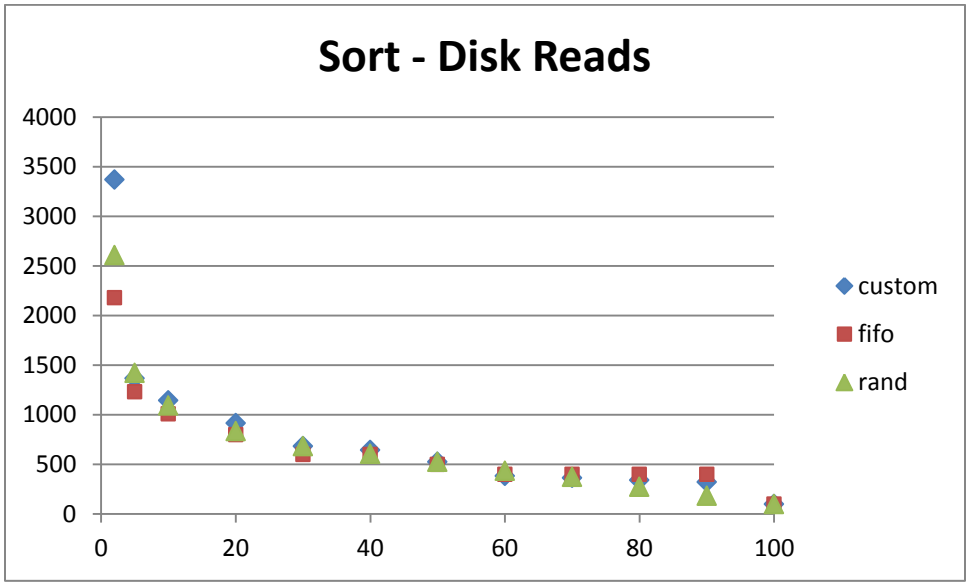


The random algorithm performs best in terms of disk reads/page faults on the scan program. This is due to the nature of the scan program, which iterates through the whole array of data 10 times. This does not have locality. FIFO performs very poorly because it is continually having the update the pages (it is trailing behind what is needed).

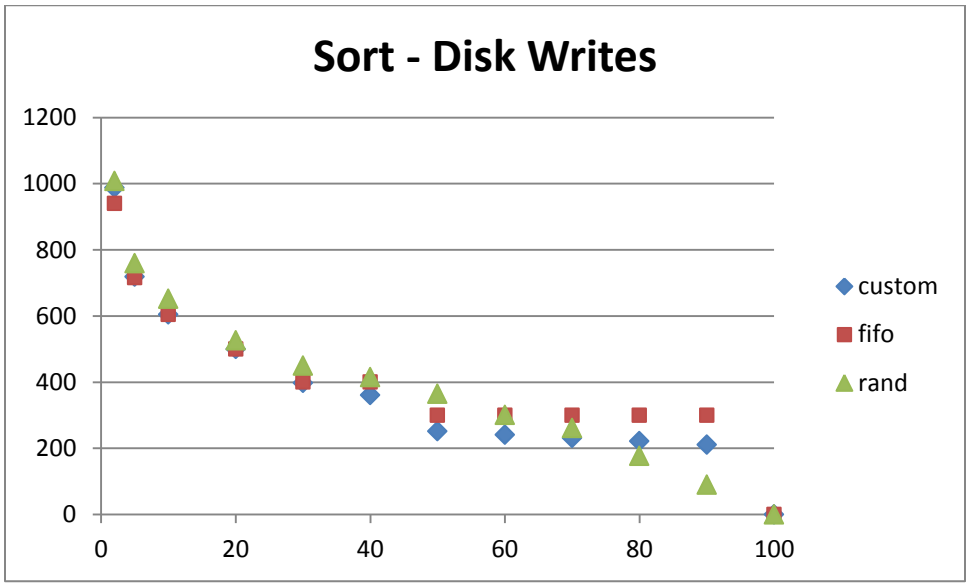


Custom performs best in terms of disk writes, but only at a large number of frames. Each of the algorithms performs poorly before this because there are not any choices of pages that have not been written to because of how the scan program executes.

Sort Program Results



The algorithms perform similarly in terms of disk reads/page faults for the sort program. With the quicksort algorithm that is used for this program finding and using a pivot in the sort can result in a large amount of jumping around in memory. Thus, each of the algorithms performs similarly because none well predict what pages will be needed in the future.



The algorithms again perform similarly with sort for disk writes. The custom algorithm is slightly better for middle-range number of frames, and rand is better at a high number of frames. Custom performs better because of the priority in selecting victims that don't have to be written back, but because this is only part of the selection, rand performs better at higher number of frames.