Project 5 lab report
Grace Bushong, Erin Turley

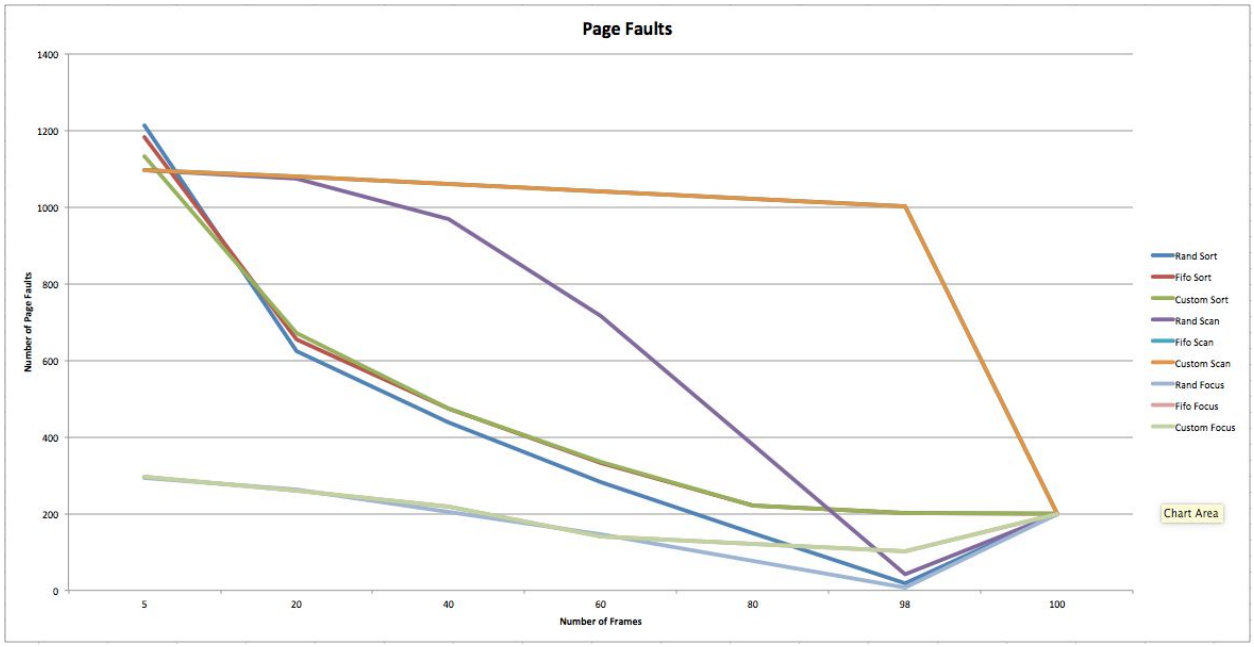
Experiment Purpose and Setup

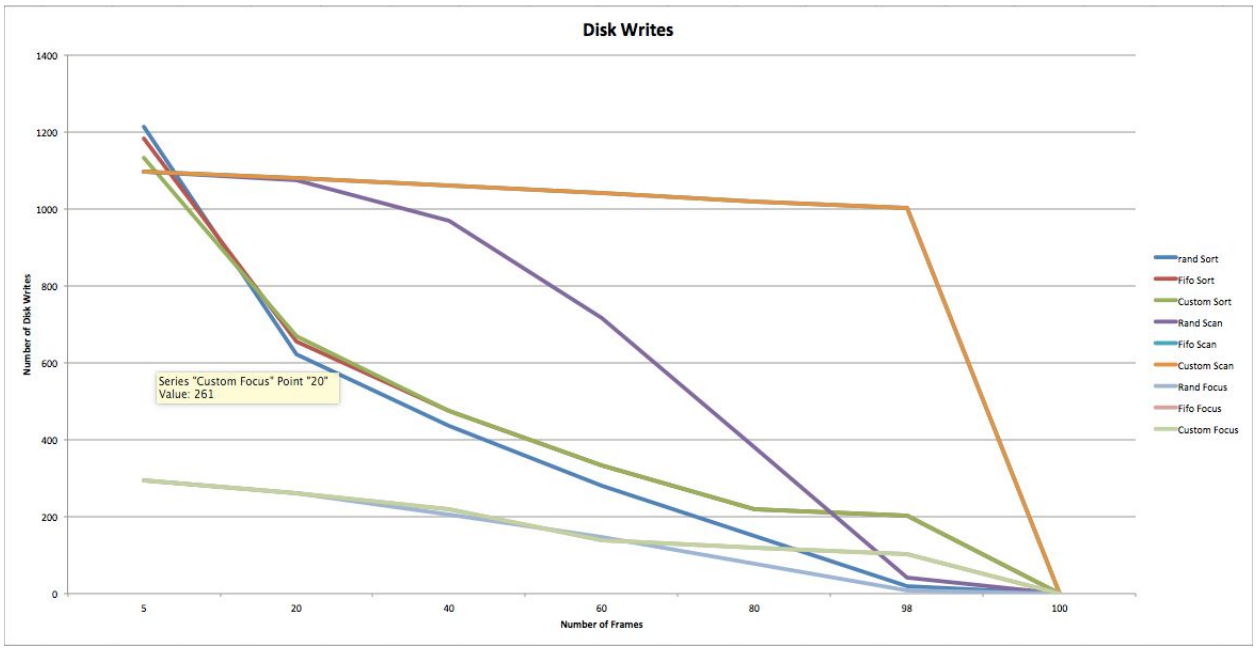
The purpose of the experiments was to determine which algorithm is the most efficient - if there is one that is more efficient overall, or which ones are better under specific conditions. The experiments were run on each combination of the algorithms and the built in programs for 100 pages and about intervals of 20 up to 100 frames. The test was run on 5 frames as the minimum and was run on 98 frames to show what the maximum looks like in terms of the way the page faults, disk reads and writes look under the normal curve, and 100 was graphed too, but it does not fit the curve because when there are the same number of frames and pages we handled it as a special case. The tests were run on student02 and the command line arguments were used as specified: ex. ./virtmem 100 20 rand sort etc. In our counting of page faults, page faults caused by incorrect permission bits were ignored, as specified by Professor Striegel.

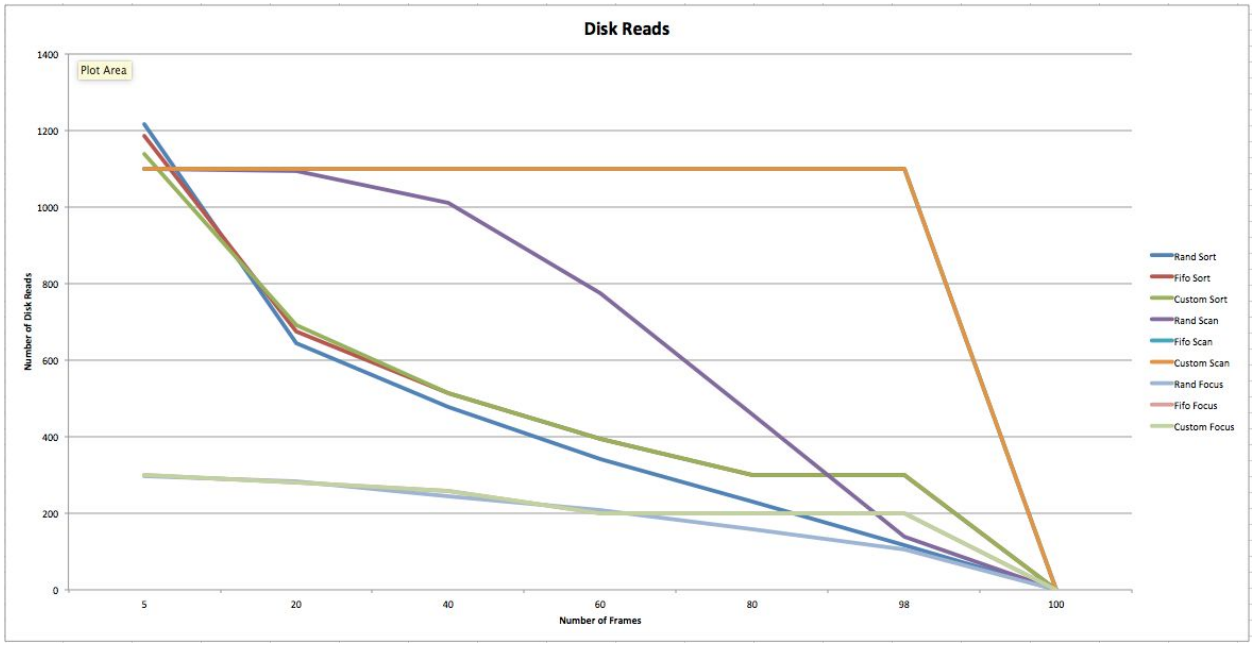
Custom Page Replacement Algorithm

Our page replacement algorithm is a variation of the Least Recently Used algorithm. However, a challenge arose when we realized that we did not get a callback on a successful read or write-- instead, we only got a callback on a page fault. With this in mind, we designed our algorithm so that the page that had least recently had its read or write bits set would be replaced in the case of a full frame table. We created an array that was the same size as the array that kept track of the elements inside the frame table. If we got a page fault, our handler first checked to see if the page was already in our frame table-- if it was not in the frame table and our frame table was not yet full, we add the page to an empty spot and set that page's entry in the custom array to zero, then increment every other element in the array by one. This custom array was meant to represent which page had most recently had its "read" or "write" permission bits set. If the page was not in our frame table and the frame table was full, we passed the page to our custom handler. However, if it was in the frame table, but had page faulted as a result of the wrong permission bits, we made that page's entry in our custom array equal to zero, then incremented every other element in the array.

If the page got sent to the custom handler, then this was a result of a full frame table, so the page would have to replace an element already in the frame table. This element was chosen by iterating through the custom array and finding the max value. The max value signified the number that had least recently had its read or write permission bits set.


Page Faults
Number of Page Faults
1400
1200
1000
800
600
400
200
0
5
20
40
60
80
98
100
Number of Frames
Rand Sort
Fifo Sort
Custom Sort
Rand Scan
Fifo Scan
Custom Scan
Rand Focus
Fifo Focus
Custom Focus
Chart Area


Disk Writes
Number of Disk Writes
1400
1200
1000
800
600
400
200
0
5
20
40
60
80
98
100
Number of Frames
Series "Custom Focus" Point "20"
Value: 261
rand Sort
Fifo Sort
Custom Sort
Rand Scan
Fifo Scan
Custom Scan
Rand Focus
Fifo Focus
Custom Focus



Explanation of Results

The different algorithms have drastically different shapes of their curves, but each algorithm has a similar shape of the curve graphing Number of Frames vs. Page Faults, Disk Reads, and Disk Writes. All of the algorithms have the same value for 100 frames, since our program creates a special case for when the number of pages is equal to the number of frames. All algorithms then have 100 page faults with 0 disk reads and 0 disk writes because they do not have to do any replacement and can simply load all the data into the physical memory once. Since all the data is in the physical memory, it will never be not found and cause a page fault. On the graphs, the Fifo Scan and Fifo Focus lines are not visible because they look exactly the same as the Custom Scan and Custom Focus graphs. The Fifo Sort is very similar to Custom Sort, but they are slightly different for lower numbers of frames - Custom is better for the lowest values, then Fifo is better for slightly higher low values, and then they become the same. Because our Custom algorithm only kept track of when frames in physical memory had permissions edited (rather than keeping track of a successful read), it was very similar to the fifo algorithm because the permission editing didn't occur that often and for the most part, the first data put in would also be the oldest data referenced and be the first booted out. Custom and Fifo performed much worse for scan than Rand did - the nature of scan is that it writes all of the data and then reads it all back, so it isn't accessing recently used data ever (where fifo and custom are stronger algorithms), but sometimes rand will perform well just by chance. The sort algorithm uses qsort, so it jumps around a lot and rand actually performs the best for that program as well. Custom and Fifo perform better than rand for low frame numbers of Focus, somewhat by chance, because the focus program depends a lot on randomly retrieved values.