Michael Burke, William Theisen

7 April 2017

Project 5

Purpose

The purpose of the project is to understand how virtual memory management works and to create a few page replacement algorithms. As a group, we wrote a page fault handler that moves data from disk onto physical memory or sets the appropriate read and write bits when a page fault occurs. We ran the experiment of student machine 03 and the command line arguments used to test were:

Random:

./virtmem 100 3 rand focus

./virtmem 100 10 rand focus

…

./virtmem 100 100 rand focus

Custom:

./virtmem 100 3 custom focus

./virtmem 100 10 custom focus

…

./virtmem 100 100 custom focus

FIFO:

./virtmem 100 3 fifo focus

./virtmem 100 10 fifo focus

…

./virtmem 100 100 fifo focus

These arguments were changed to handle scan and sort as well.

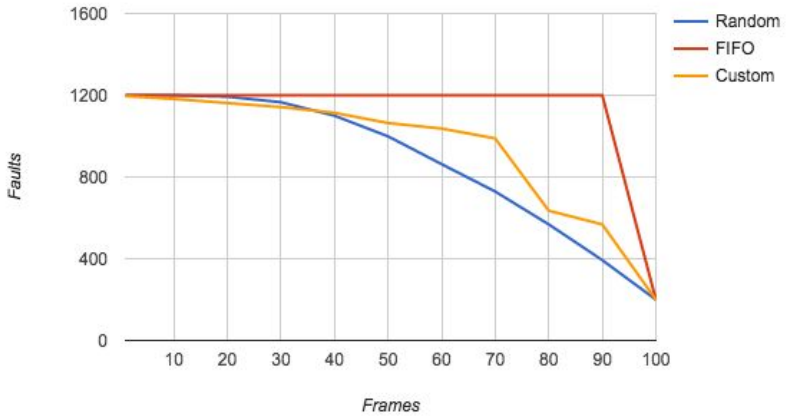
Custom Algorithm

Our custom algorithm is based off of the least recently used model, with a longevity boost for files that have been written to. Using a struct called timeval, once a page is accessed, we store a timestamp corresponding to the access time to an array called accessTable->elements[]. This timestamp struct has two values in it: seconds and its remaining microseconds. Pages that are not currently loaded into physical memory have their written bit set to 0.

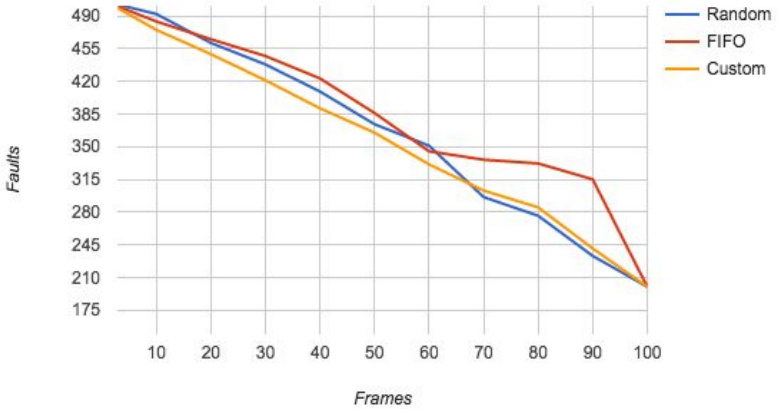
When replacement occurs, we iterate over the accessTable->elements[] finding the minimum value which represents the oldest written page. We compare each element's timestamp second values. If the seconds equal each other, we then compare the microseconds. We give a priority boost to recently written pages. We add 10,000 microseconds to these pages' timestamp every time the bit is changed to write. Handling this addition is tricky. When we add 10,000 to the microseconds variable, sometimes the resultant variable is greater than or equal to one second. When this occurs, we need to instead add one to the seconds variable and then subtract 900,000 from microseconds variable which is a total time increase of 10,000 microseconds.

Results

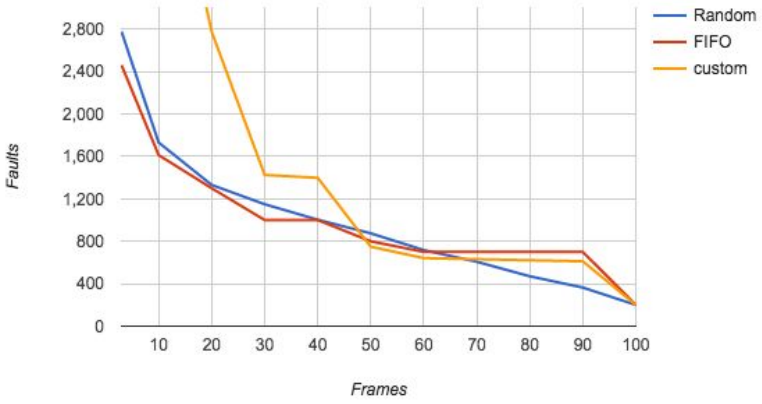
Scan Faults



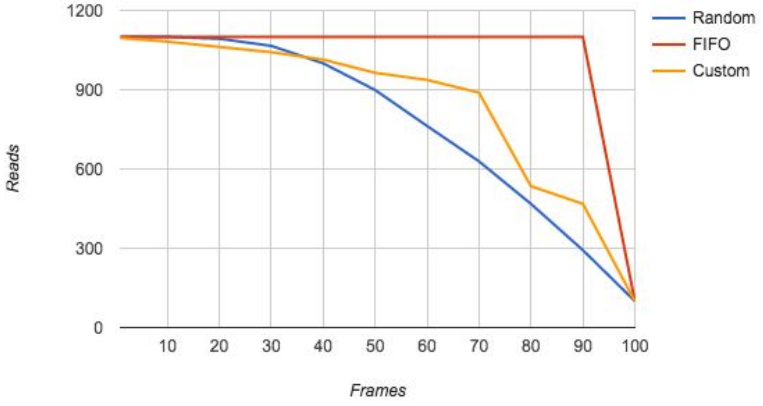
Focus Faults



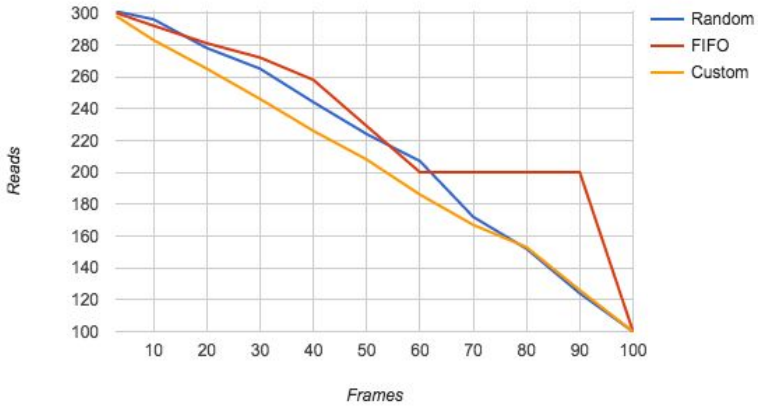
Sort Faults



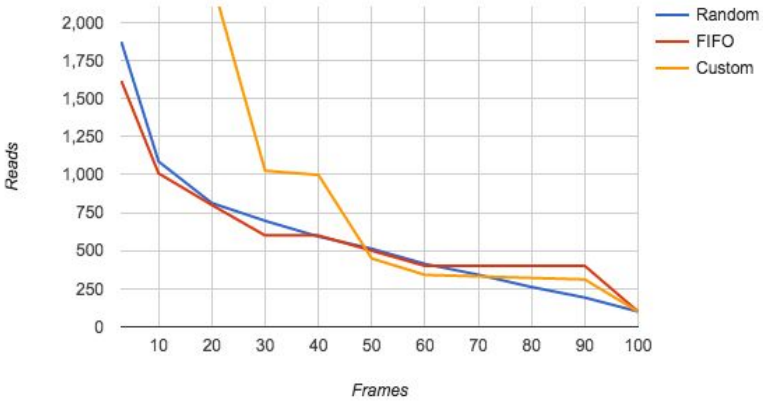
Scan Reads



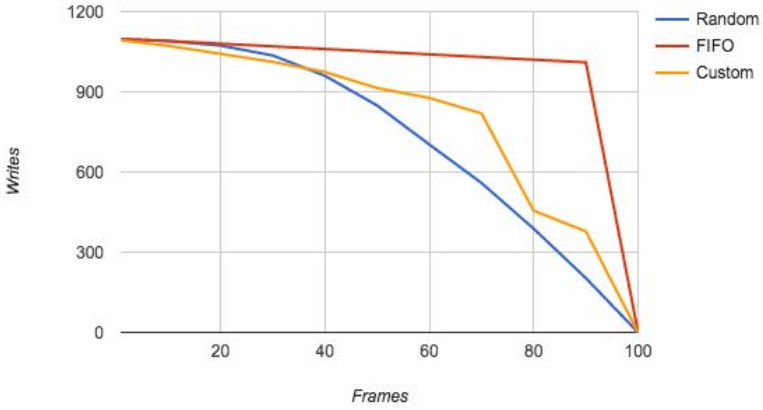
Focus Reads



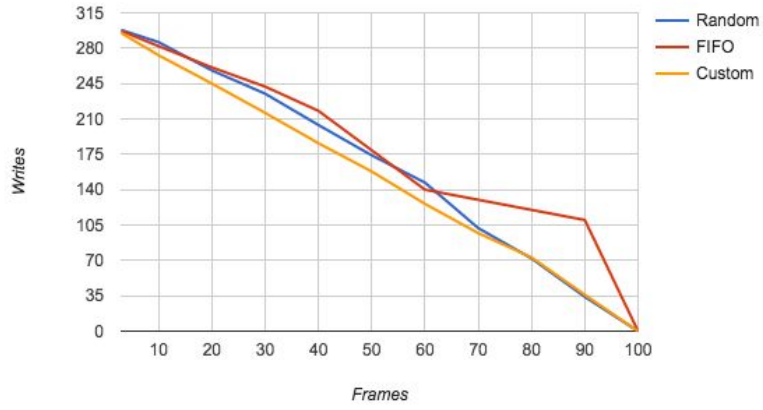
Sort Reads



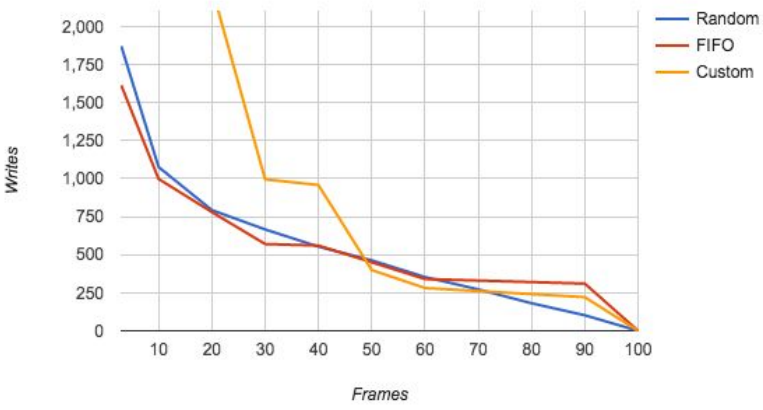
Scan Writes



Focus Writes



Sort Writes



Analysis

For the program focus, all three algorithms performed at about the same rate, with our custom algorithm slightly beating random and FIFO in faults, reads, and writes. Custom's technique of replacing the least recently used page (with a boost to written pages) must give it a slight advantage against random and FIFO.

For the scan program, random performs the best while FIFO performs the worst. It is likely that due to the looping access pattern in scan FIFO is evicting exactly the next page it needs as it goes along, leading to this terrible performance.

For the sort program, custom performs very very poorly with less than 10 or 20 frames. Once the frames equal 30 or more, custom performs just as well as random and FIFO. The sort the quick sort uses is a quicksort. The branching of quicksort's recursion or the divide and conquer nature of the program must make custom very inefficient with such little frames as 10 and 20.

FIFO has an intrinsic flaw in its logic where it is inefficient in programs that in order access data. FIFO removes the page that was loaded into memory first, but if a program loops its memory access patterns fifo will always remove the next page it needs as it goes back along the loop. Rand and custom don't necessarily have this behaviour (they could but it is likely they do not) and this may be why FIFO's performance lags behind random and custom against all three programs as a cyclical access pattern of memory is very common. It appears that out of the three above random appears to return the best, consistent results. However that being said, if you knew ahead of time the value of all the variables it might be possible to pick a better algorithm for the situation. No one algorithm is better in every situation which explains both why there are so many different algorithms, and why there is still development in the field. The number of reads and writes is also important as the type of hardware could cause the cost of

one of these algorithms to be much higher than the other and thus you would want to choose an algorithm tailored to the specific situation.