CSci 5103 Advanced Operating Systems Programming Assignment 3 Report

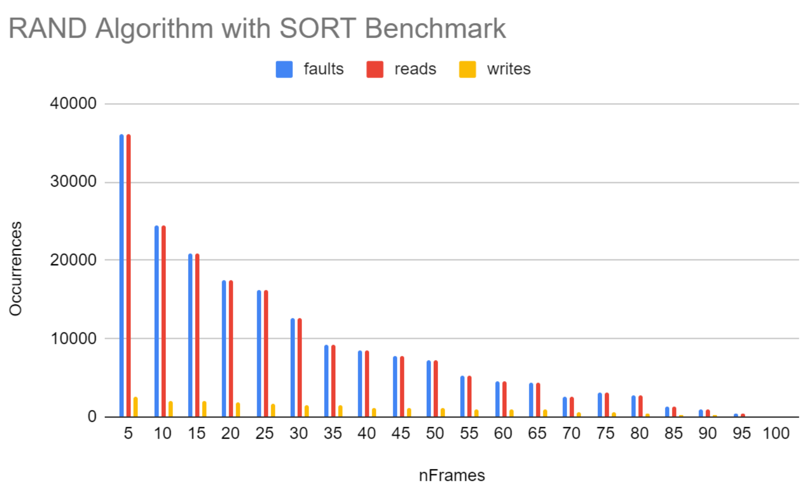
Chase Anderson *and08479*
Emir Sahbegovic sahbe001

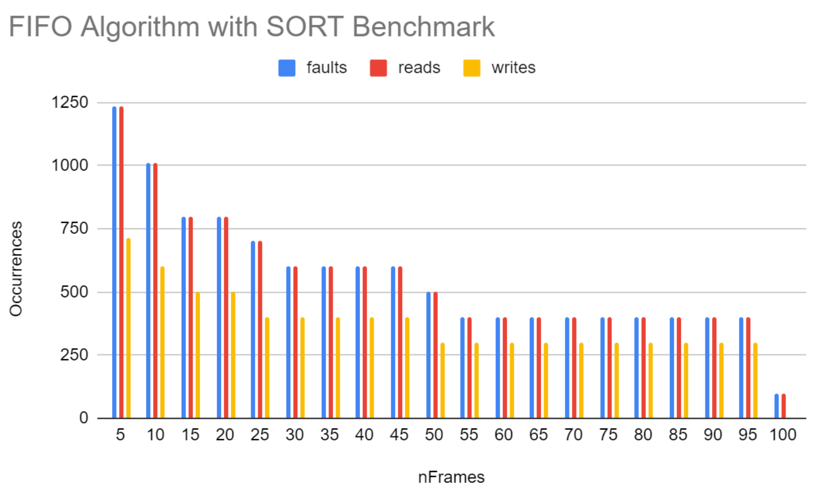
Introduction

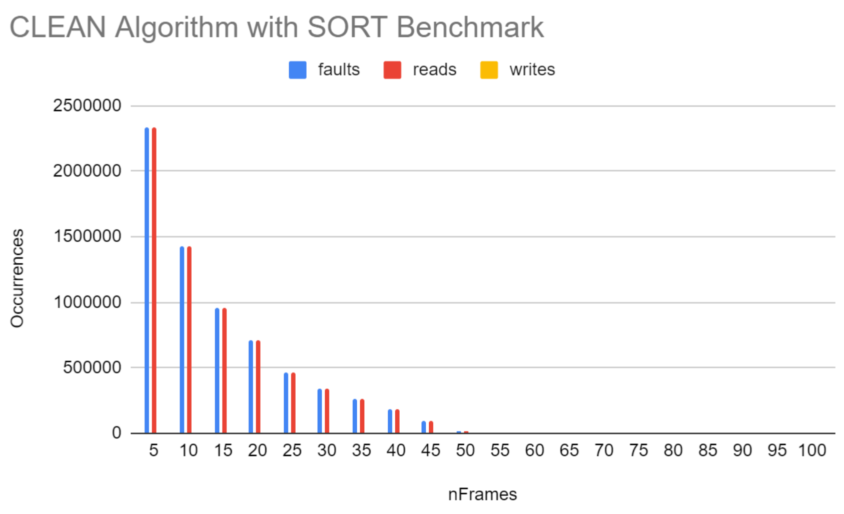
In this experiment, the effect of different page replacement algorithms were analyzed across three different task programs. The purpose is to measure the occurrences of page faults, disk reads, and disk writes and visualize how different replacement algorithms affect performance. One algorithm being a random victim selection, another being a basic FIFO replacement, and finally we made a customized replacement algorithm that uses the modified (dirty) bit to determine if a certain page is meant for replacement. Then on, we have three tasks to test these algorithms on. First, a sort program using an implementation of quicksort. Next a scan program with mostly sequential memory access. Then a focus program which randomly selects portions of the input data to perform manipulations. We have data and graphs to showcase the performance and patterns associated with each respective algorithm and task.

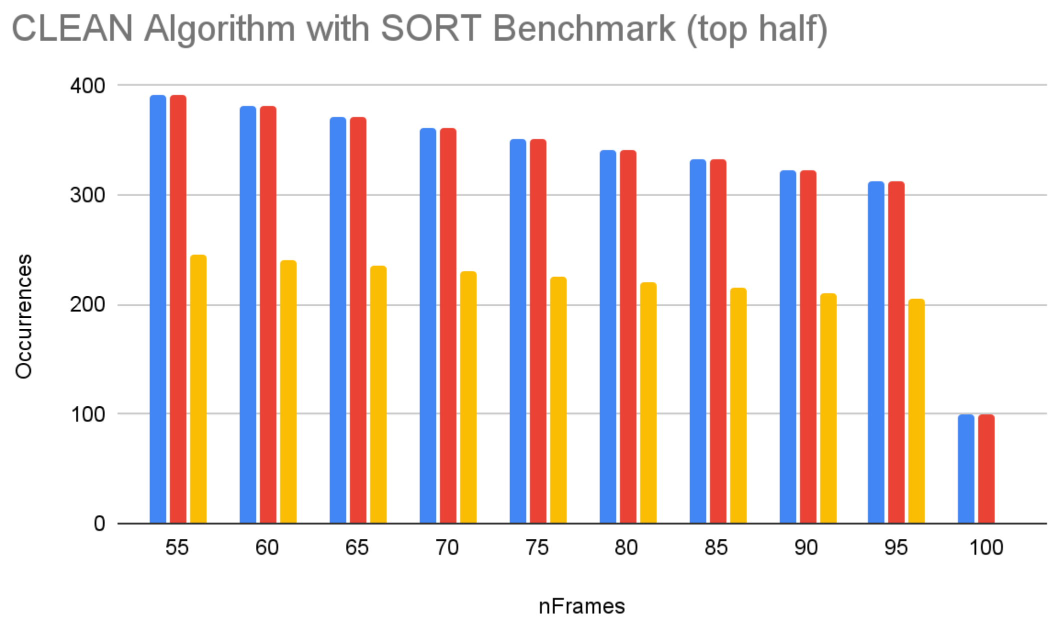
We ran the experiments by establishing an SSH connection into a lab machine (*csel.kh1250-05.cselabs.umn.edu*) via VS Code. We ran the code using the commands provided in the README.txt. When running the command, nPages would always be 100, and we would use different values for nFrame. We started with an nFrames of 100 and decremented by 5 each time to collect data.

Figures


RAND Algorithm with SORT Benchmark
faults
reads
writes
Occurrences
nFrames
40000
30000
20000
10000
0
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

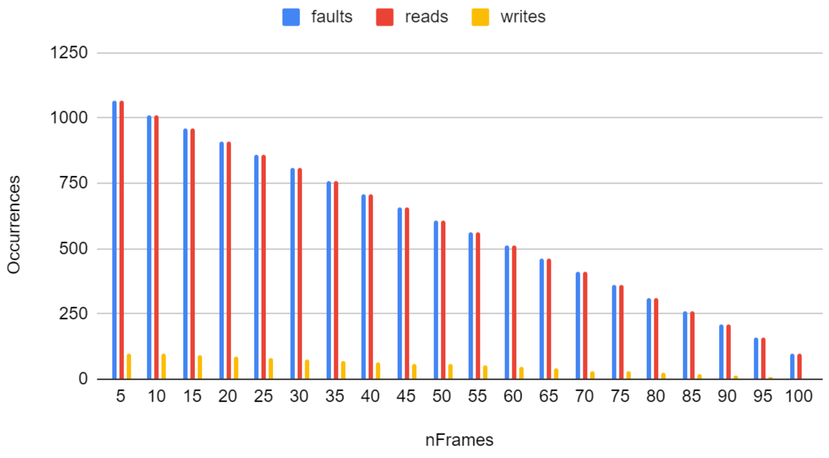

FIFO Algorithm with SORT Benchmark
faults
reads
writes
Occurrences
nFrames
1250
1000
750
500
250
0
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100


CLEAN Algorithm with SORT Benchmark
faults
reads
writes
Occurrences
nFrames
2500000
2000000
1500000
1000000
500000
0
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

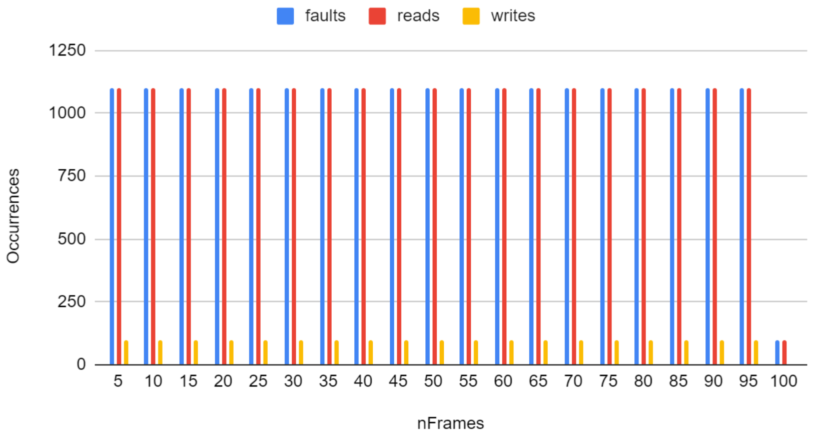


CLEAN Algorithm with SORT Benchmark (top half)
Occurrences
400
300
200
100
0
55
60
65
70
75
80
85
90
95
100
nFrames

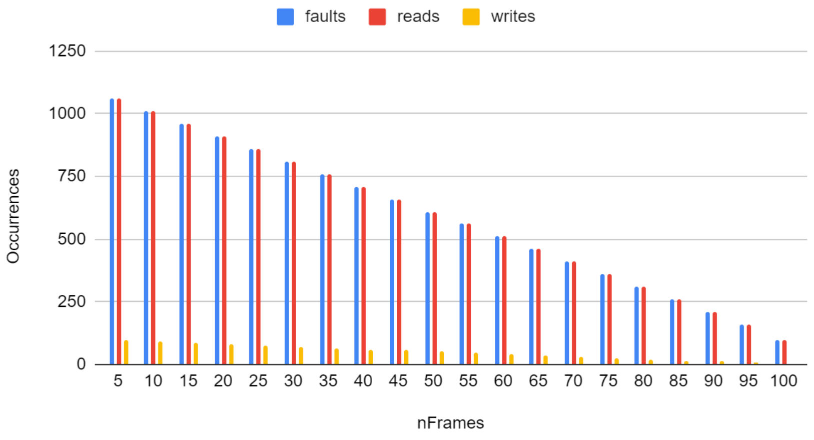
RAND Algorithm with SCAN Benchmark

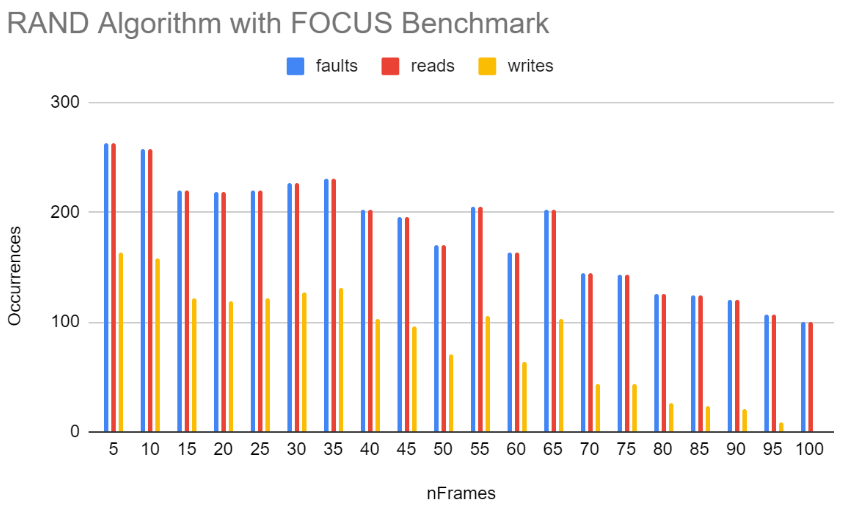


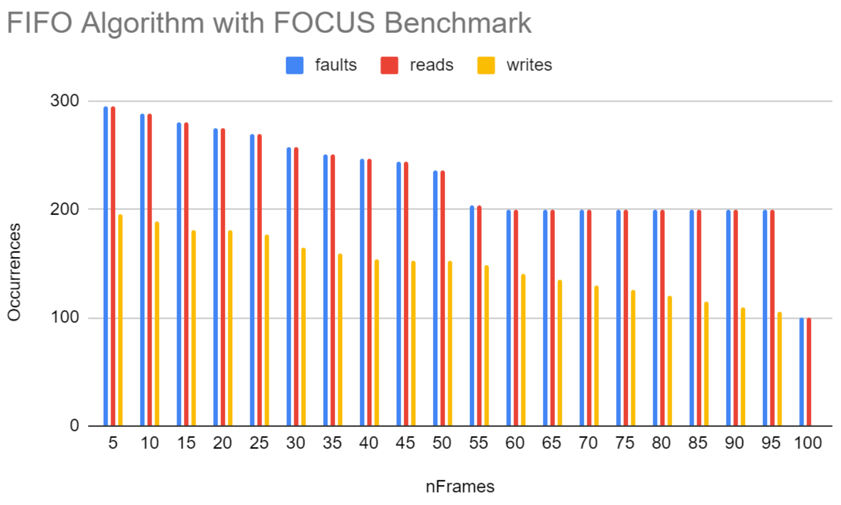
FIFO Algorithm with SCAN Benchmark

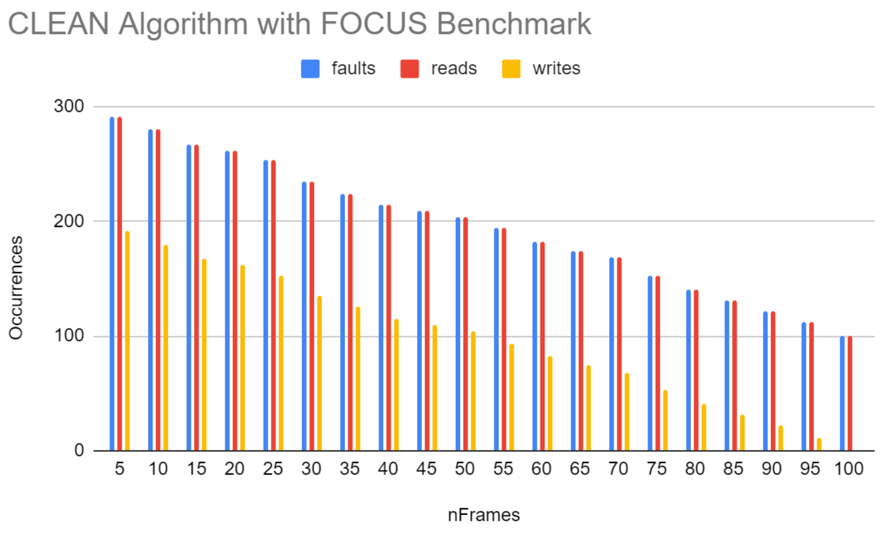


CLEAN Algorithm with SCAN Benchmark




RAND Algorithm with FOCUS Benchmark
faults
reads
writes
Occurrences
nFrames
0
100
200
300
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100


FIFO Algorithm with FOCUS Benchmark
faults
reads
writes
Occurrences
nFrames
0
100
200
300
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100


CLEAN Algorithm with FOCUS Benchmark
faults
reads
writes
Occurrences
nFrames
0
100
200
300
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

Analysis

SORT ANALYSIS

Starting with page faults, the basic FIFO had significant improvements upon the results of our random test. Where sort, and namely quicksort, is using a recursive partitioning of the input data to sort through, the queue structure used in FIFO seems to perform well. Now in comparison to our customized clean (dirty bit averse) algorithm. We see that there exists extremely poor performance for nFrames less than half of nPages. However, once the 50% threshold is passed, we see that the clean algorithm has better page fault performance than the basic fifo algorithm. Due to the nature of our control logic, our disk reads are directly inline with the page faults, for each page fault will cause a value from the secondary disk to be loaded into physical memory.

Now, the rand algorithm had a significantly larger number of disk writes compared to both fifo and clean. We assume the deficit in rand is due to each partition in the quick sort algorithm being poorly managed through random replacement. The consistency of the fifo algorithm as nPages and nFrames approach each other can be explained by being able to accommodate the queued frames as nFrames get larger approaching 100. As for the clean algorithm, we can see the same pattern. The writes become lower as the nFrames become more manageable. We can also see the dirty bit control logic assist in the writes with every increment of nFrames. This is similar to the rationale in the textbook for the enhanced second chance algorithm, where the dirty bit and reference bit assist in the performance especially in the case of eviction.

In the case where we have a task requiring the behavior of sorting like this, we would elect to use the clean fifo. This is assuming we have the liberty of having more than half the number of pages for frames, in hopes to avoid the ill behavior seen in our data.

SCAN ANALYSIS

Our results for the scan program show that FIFO had arguably the worst performance, having the most disk reads and disk writes for any given number of frames. This makes sense because FIFO indiscriminately replaces pages during execution

The CLEAN algorithm has less disk writes, which makes sense because it prioritizes replacing the pages which have read only bits. Because all of the reading is done at the end of the scan program, the CLEAN algorithm will replace one of the read-write pages and keep replacing that page when page faults occur, leaving the other pages alone. We also found that the amount of page faults and disk reads decreased by 50 for every 5 frames we add to the physical memory. We believe that this is due to line 112 of the scan program. The reading portion of the scan program is repeated exactly 10 times, which means that there will be 50 less page faults when we increase the amount of frames by 5. This is because all but one of the pages were untouched during the reading section, so these pages will not fault when they are read from. Arguably, CLEAN has the best performance, having the least reads and writes.

The results of the RAND algorithm are strange; they seem to be exactly the same as the CLEAN algorithm. We also noticed that the results of the RAND algorithm do not vary every time the program is executed. We believe that this may be due to how we seeded the random number generator, but we still don't fully understand these results. The scan program is the only program where the RAND algorithm has this kind of behavior, so we know that it must have something to do with the nature of the scan program. At the moment, we believe that it has to do with the looping part of the reading portion, similar to the CLEAN algorithm.

FOCUS ANALYSIS

In this algorithm, we see that the worse page fault performance comes out to be the basic fifo. As expected, we still see the decrease in faults, reads, and writes using the clean algorithm. The focus program repeatedly takes random portions of data, then indexes, using stack specific randomization, to write a random byte into a zero-initialized array. Using this rand() function points to the random page replacement algorithm for being the best choice. Looking at the data, we can see that the rand algorithm produced less page faults and disk writes for 80% of the inputs for nFrames. On average the rand algorithm performs well, given the nature of the focus program.

Another thing to note is that we thought that seeding the RNG in the fault handler may change the behavior of the RNG in the program, but it seems that it does not. We aren’t completely sure why that is, but we think that each stack has its own RNG.