Brad Sherman
Professor Thain

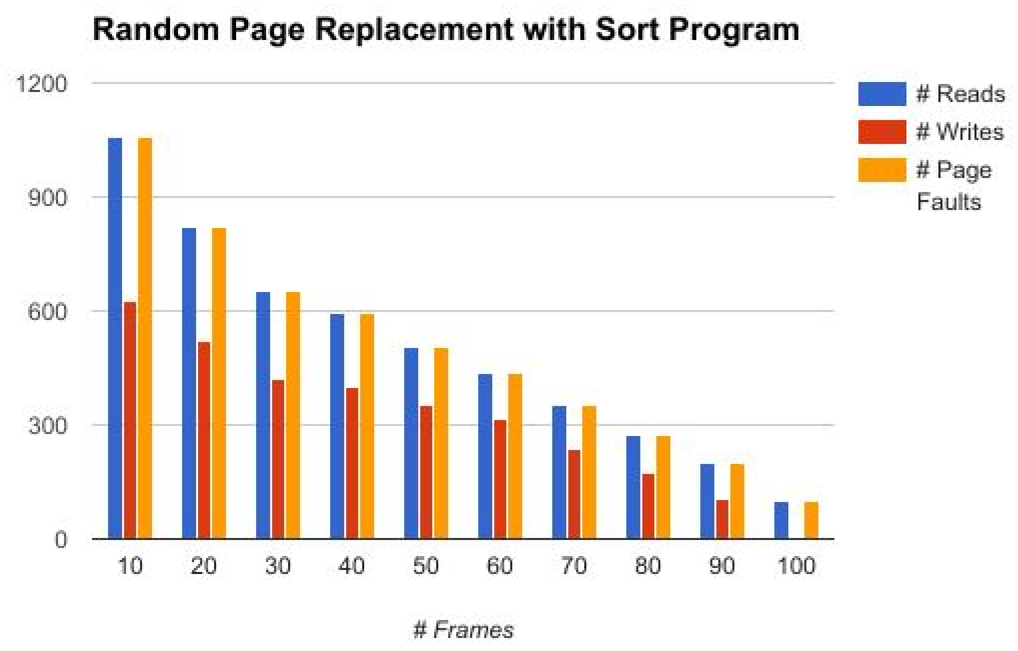
# Operating Systems - Project 5

**Purpose:**
The purpose of this experiment was to see how different page replacement algorithms affect the performance of different programs that access disk differently. The page replacement algorithm has a huge impact on the performance of a program that accesses disk a lot. Since disk reads and write are very expensive, we want to avoid them as much as we can. So in this project we aimed to figure out the best way to do that. I ran these experiments on my personal laptop which is a Lenovo Thinkpad W540, running Ubuntu 16.04.

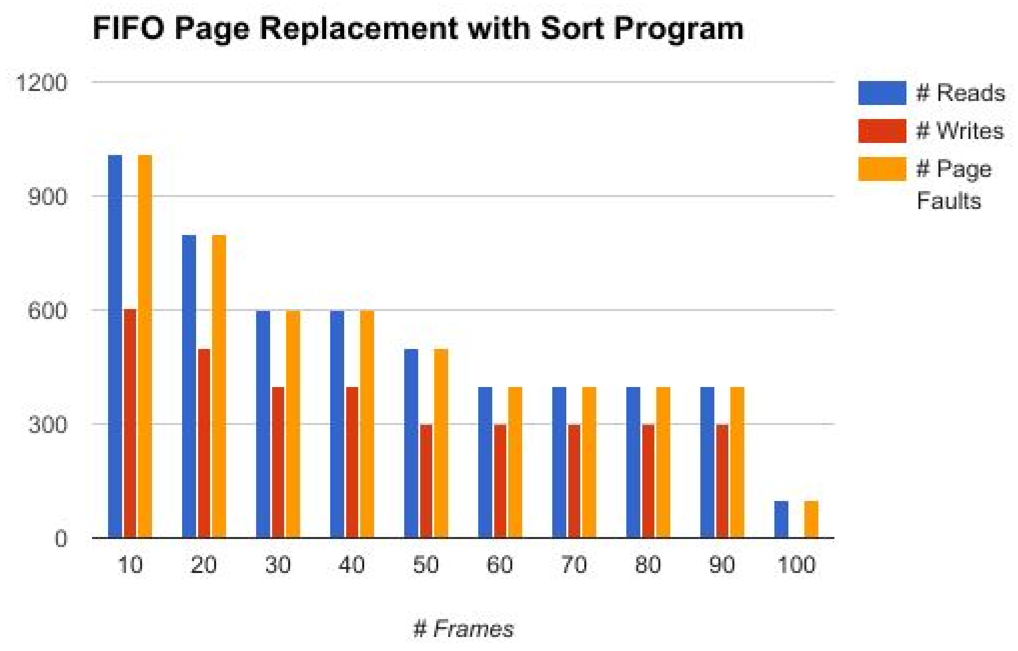
**Custom Algorithm:**
My custom page replacement algorithm is a variation of the random page replacement algorithm. Basically, it generates a random frame between 0 and nframes. Then, it gets the page that that frame corresponds to, and retrieves the permissions on that page. If this page has write permissions, we skip it, and find another page. This continues until we reach a maximum number of tries. I originally set the maximum to the number of frames, but the results were lacking, so I reduced the maximum number of tries to the number of frames divided by 3, and saw better results. I came up with 3 after a bit of playing around with the algorithm and deciding which reduction factor gave the best results. The basis of this algorithm is that, when we kick out a frame that corresponds to a page with write permissions, that means that that page is dirty, so we must write back before we kick it out. This turns out to be costly, so we want to avoid that. So whenever possible, we pick a page that does not have the write permissions set, so we only have to read in the new page. If the maximum number of tries is met and we have not found a page with no write permissions, we just return the last random number generated, so as not to spend too much time trying to find the perfect page.

**Results:**
Below are a series of charts that I made from the results of running my program with 100 pages and 10-100 frames in increments of 10 for each program and page replacement algorithm.

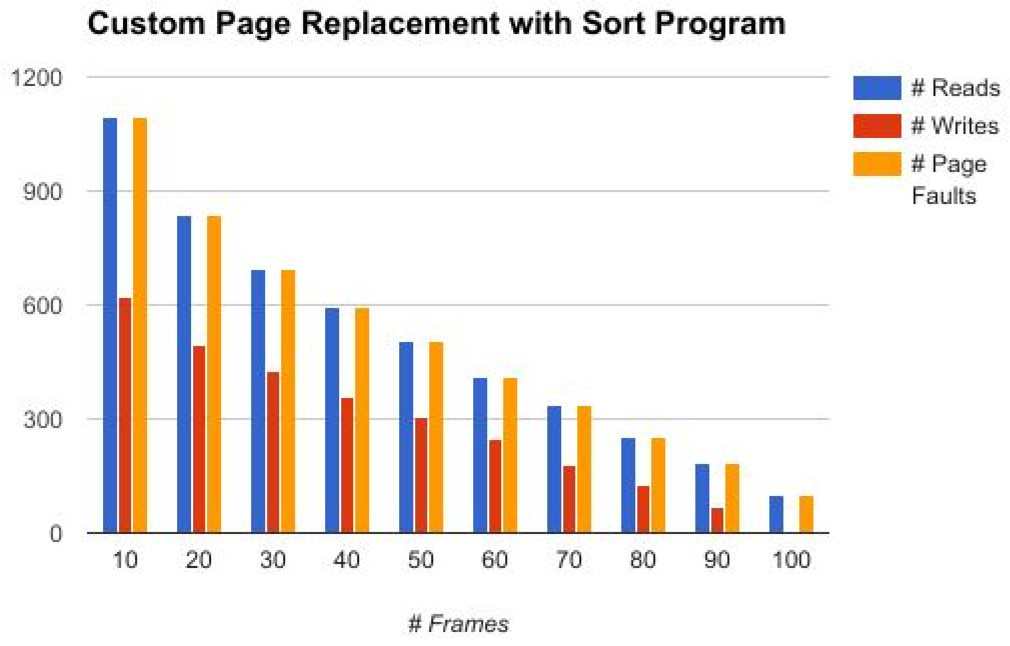
## SORT



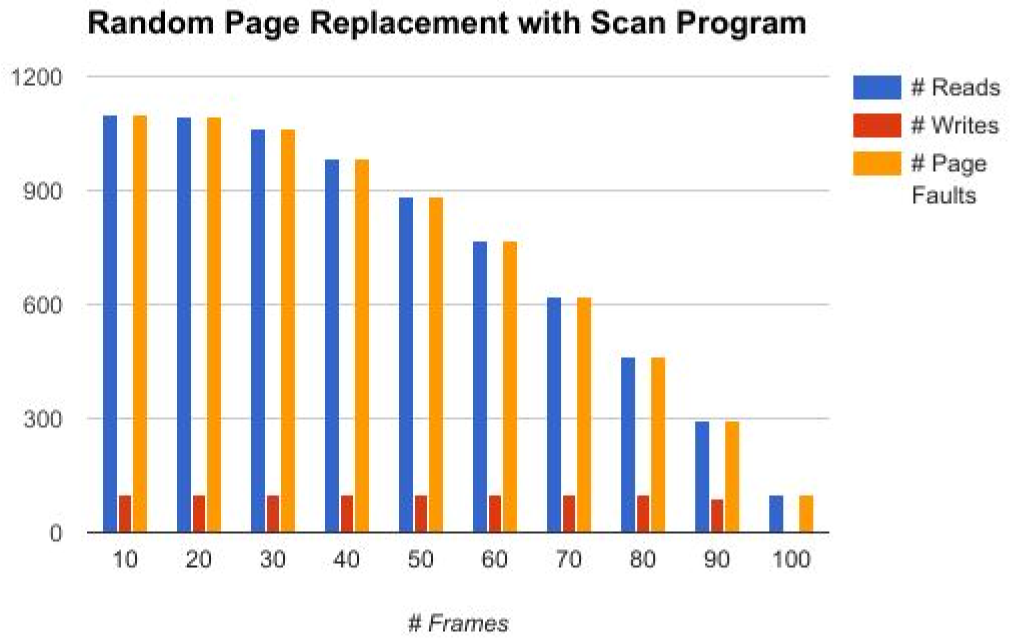
Random Page Replacement with Sort Program
1200
900
600
300
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults



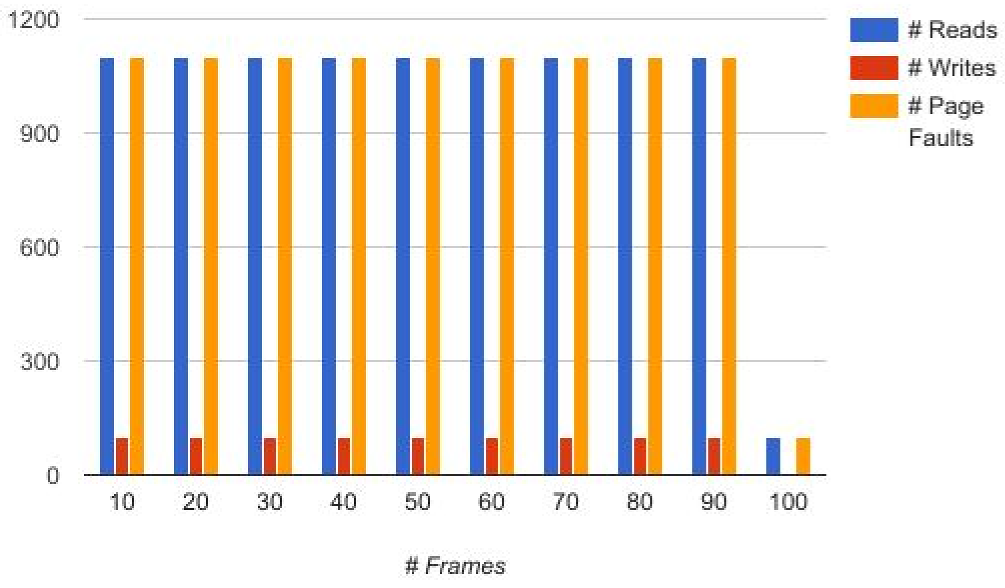
FIFO Page Replacement with Sort Program
1200
900
600
300
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults


Custom Page Replacement with Sort Program
1200
900
600
300
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults

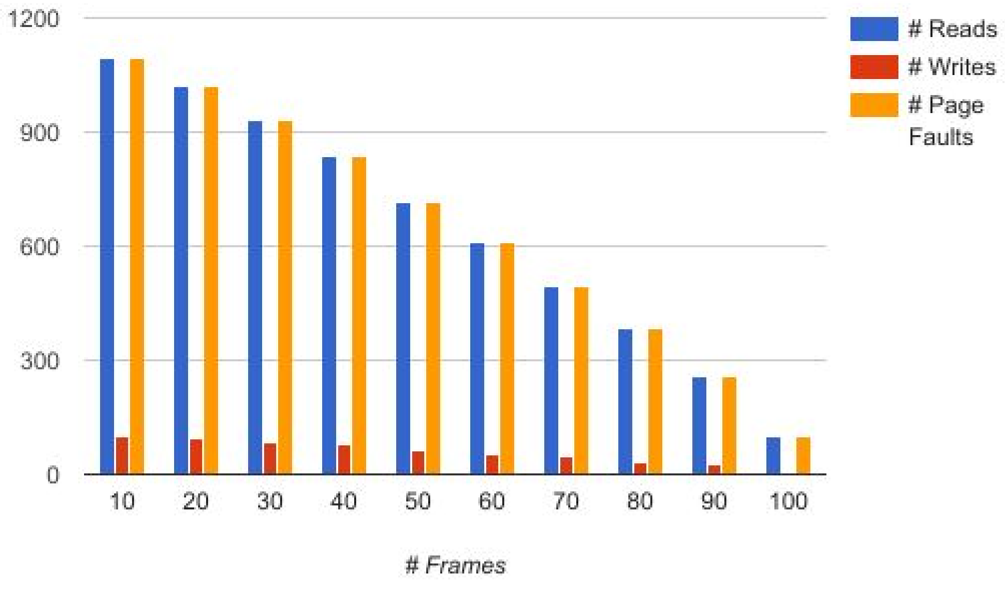
**SCAN**


Random Page Replacement with Scan Program
1200
900
600
300
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults

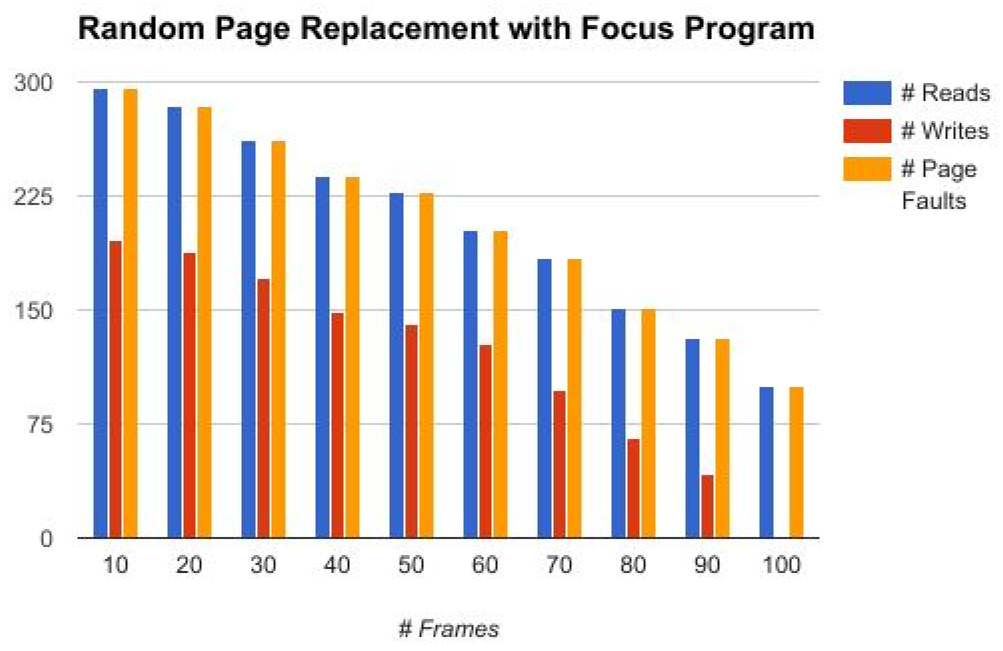
FIFO Page Replacement with Scan Program

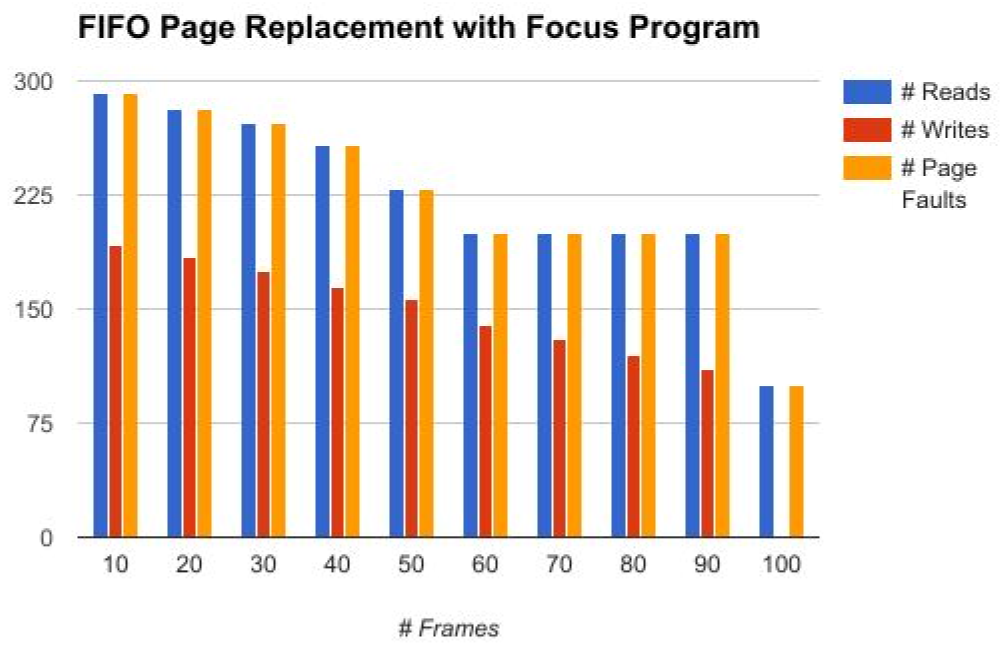


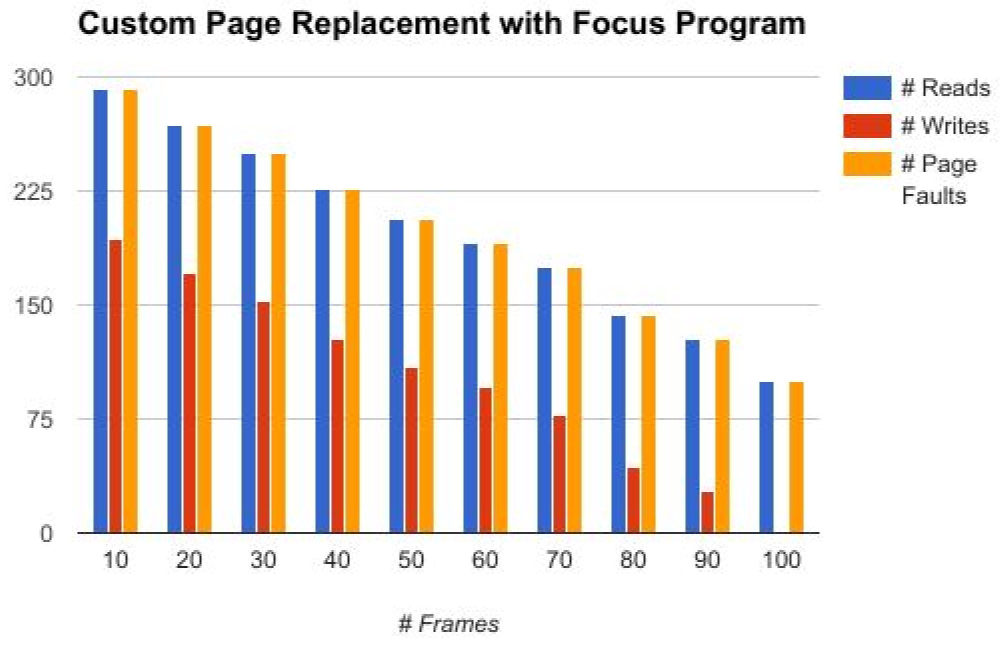
Custom Page Replacement with Scan Program



**FOCUS**


Random Page Replacement with Focus Program
300
225
150
75
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults


FIFO Page Replacement with Focus Program
300
225
150
75
0
10
20
30
40
50
60
70
80
90
100
# Frames
# Reads
# Writes
# Page Faults



**Discussion:**
As a general note, all three algorithms tend to decrease in the number of reads, writes, and page faults for each program as the frame number increases. This makes sense because if we have more frames, then there will be less times that the page we want is not in memory and therefore less times we have to read/write from disk. It is hard to make a general statement about the algorithms since it varies with regards to the program because of the way each program accesses memory, so I will compare the performance of each algorithm in regards to each program. For sort, all three algorithms start with about the same number of disk reads/writes and page faults, and decrease as the frames increase. However, fifo tapers off and rand and the custom algorithm outperform fifo with a higher ratio of frames to pages. This is likely due to the fact that as the number of frames increases, there are more better candidates for eviction, but since fifo has a fixed pattern it does not find these desired pages, whereas rand and custom find them more often. For scan, fifo is by far the worst algorithm because it does not trend downwards as the number of frames increases like rand and custom do. This is definitely due to the access pattern that scan induces, which counteracts the fifo pattern of page eviction. Both rand and custom trend downwards as expected, with custom having a slightly greater rate of decrease in the number of disk reads/writes and page faults. For focus, once again, all programs trend downwards, but fifo tapers off towards the end, and rand and custom are better with a higher number of frames. Overall, fifo performed worse than rand and custom as the number of frames increased. This is clearly shown in the charts above. Overall, the experiment succeeded in showing the strengths and weaknesses of two well known page replacement algorithms, and another custom algorithm I made.