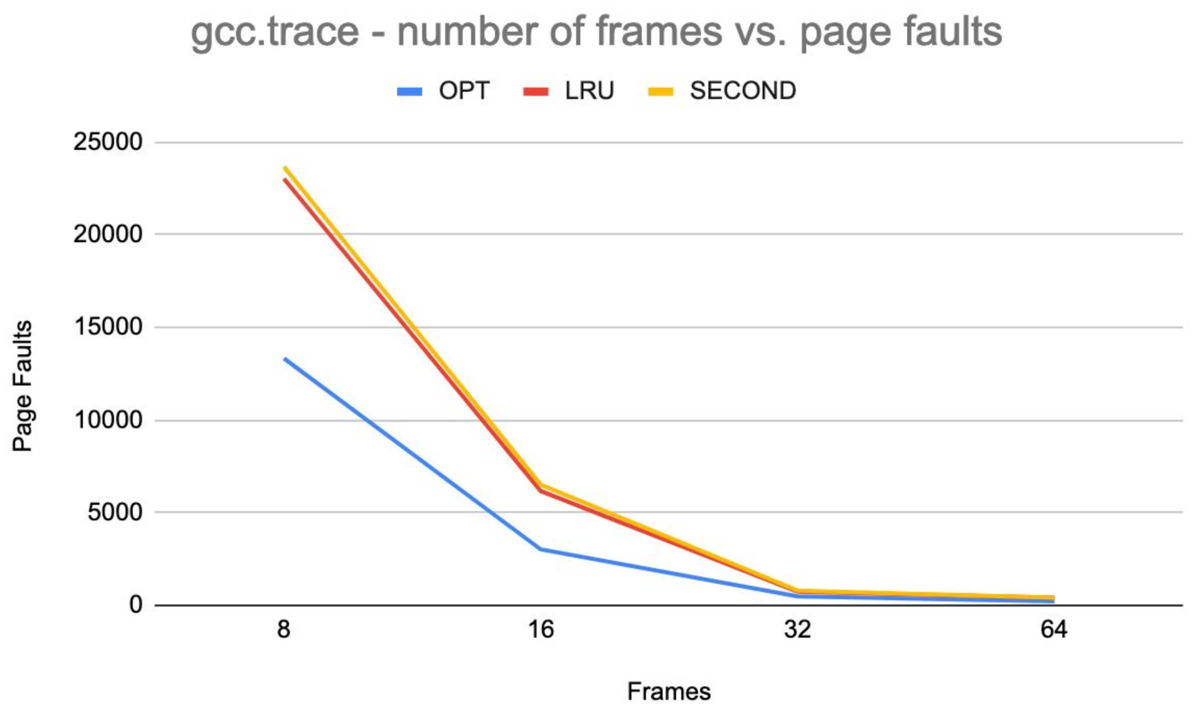
Erik Roeckel
4/6/20
CS 1550

Project 3 Writeup

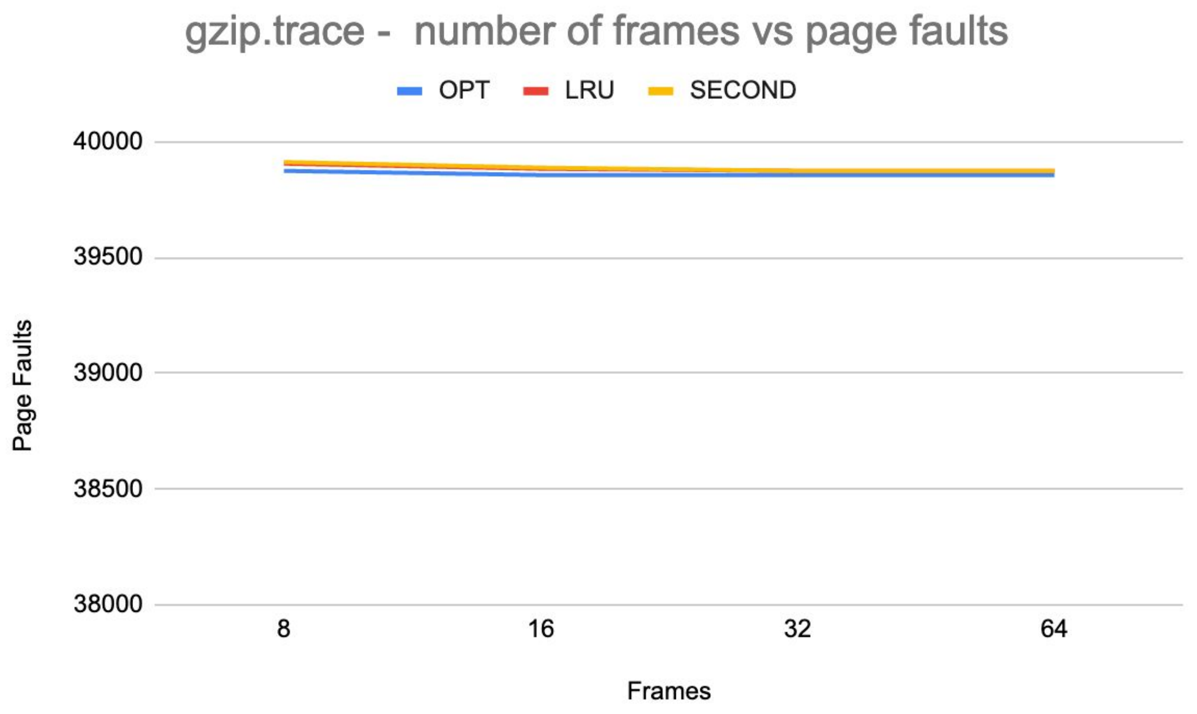
gcc.trace - page faults

# Frames	OPT	LRU	SECOND
8	13328	23027	23680
16	3020	6172	6517
32	491	736	797
64	221	409	412



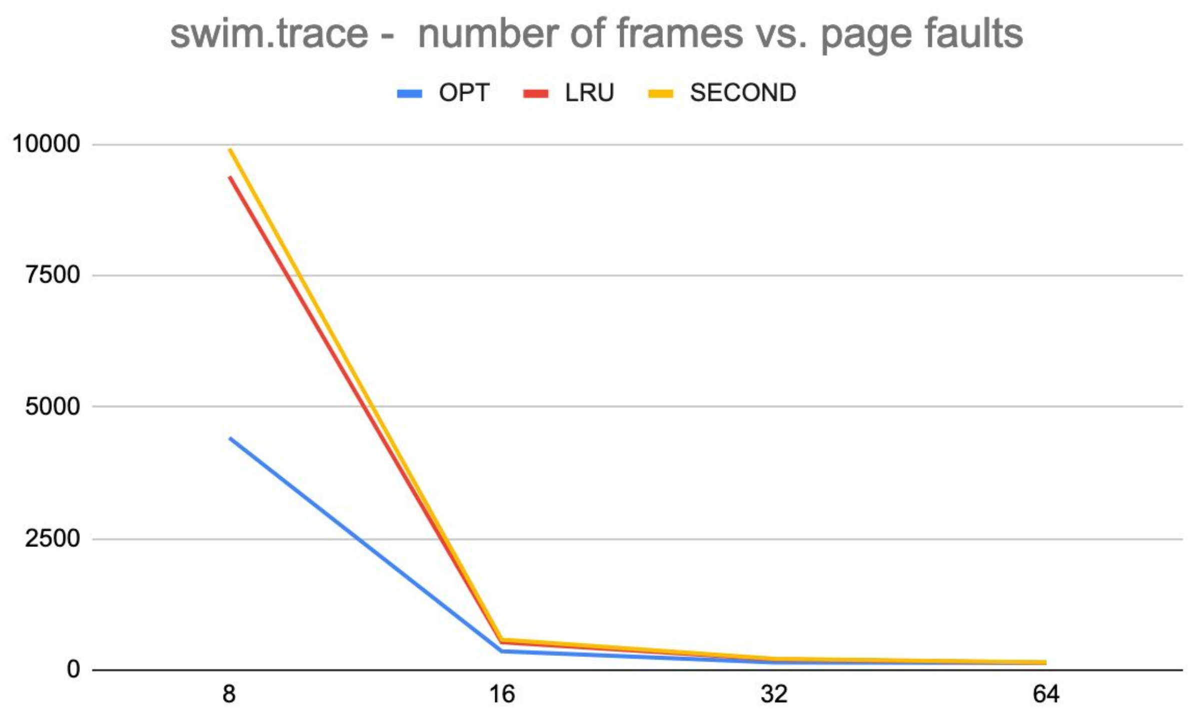
gzip.trace - page faults

# Frames	OPT	LRU	SECOND
8	39874	39905	39912
16	39856	39883	39888
32	39856	39874	39874
64	39856	39874	39874



swim.trace - page faults

# Frames	OPT	LRU	SECOND
8	4417	9387	9916
16	358	530	579
32	144	205	213
64	135	140	147



Writeup Part 1 - Best Algorithm for Real OS

Based on the results from running the three trace files on all three algorithms, I believe that LRU would be the most appropriate algorithm to use for page replacement in a real Operating System. It is obvious that OPT would be the best algorithm out of these three, however, OPT assumes perfect knowledge of all pages to be allocated in physical memory. This means that OPT is not plausible for a real operating system because a real operating system would not have this perfect knowledge. Therefore, I choose LRU as the best page replacement algorithm for a real OS. The reason I chose

LRU is that it incurs fewer page faults in all instances when compared to second chance. Being that using LRU leads to fewer page faults it can be assumed that LRU is superior to the second chance algorithm.

Second Chance - Belady's Anomaly

# Frames	**gcc.trace**	**gzip.trace**	**swim.trace**
10	17220	39897	4033
20	3691	39877	448
30	921	39875	233
40	573	39874	179
50	486	39874	154
60	422	39874	148
70	400	39874	143
80	376	39874	140
90	367	39874	138
100	355	39874	135

Writeup Part 2 - Instances of Belady's anomaly

After running the second chance algorithm on gcc.trace ranging the number of frames from 2 to 100 I found a few instances of Belady's anomaly. Belady's anomaly occurs when the number of frames in RAM are increased, yet the amount of page faults increases. Inside gcc.trace a few examples of Belady's anomaly occurred when the number of frames were increased from: 83-> 84 ,87-> 88 , 89 -> 90, 99 -> 100. After running the second chance algorithm on gzip.trace ranging the number of frames from 2 to 100 I found one instance of Belady's anomaly when the number of frames was increased from 12 -> 13. After running the second chance algorithm on swim.trace ranging the number of frames from 2 to 100 I found a few instances of Belady's anomaly when the number of frames were increased from: 70 -> 71, 75 -> 76, 76 -> 77, 83 -> 84.

OPT

Writeup Part 3 - Implementation and Runtime

To implement OPT I used an ArrayList of Integers to represent the current state of physical memory (RAM) and stored the page numbers currently in RAM in this arraylist. Because OPT assumes perfect knowledge of all memory allocations ahead of running the page replacement algorithm, I decided to pre-process all of the addresses read in from the trace file into a hashmap. For this hashmap, I used the current page numbers as keys and a linked list of integers as the values. To properly pre-process the virtual memory address into the future use hashmap for each page number, I went through and created a linked list of integers that holds all the ranks of when that page number must be allocated into memory. Once this hashmap was full of all the future allocations, if a page fault occurs and physical memory (RAM) is not full then the page is allocated to the next available slot in RAM. However, if physical memory is full then my algorithm decides on a page to evict from RAM. To decide on the next page to evict from RAM (the victim page) I then go through the hashmap of future uses and search for a page currently in RAM that either has a linked list head of null (meaning this page is never allocated again) or has the highest linked list head out of all the current pages in RAM (meaning it is used furthest in the future). After the victim page is found we remove it from RAM and replace it with the current page being allocated. The average runtime of my OPT algorithm is O(n) where n is the number of addresses to be allocated from the trace file. The reason why my algorithm has decent performance is that I utilized a hashmap to store all future uses of the page, therefore locating this page number in the hashmap has very good performance.