Marianne Fuyu Yamazaki Dorr
11/18/2021

Lab 8 Report

FIFO Implementation:

Before receiving page requests, initialize an integer variable named “placeInArray” to 0 which will keep track of the current location in the cache array.

While receiving page requests, check if the requested page is already in the cache. Create a boolean variable named “foundInCache” and set it to false. Iterate through the cache with a for loop. If the requested page is found, set “foundInCache” to true and break out of the loop. Since the page was found, there is no need to increment the total number of page faults. Proceed to the next page request.

However, if the page requested is not found, meaning “foundInCache” is still false, increment the total number of faults by one and place the page at the current index location given by “placeInArray”. Now, we need to increment “placeInArray” by one, but if we are at the end of the cache array, reset “placeInArray” to 0.

This algorithm will naturally fill out the empty slots and then replace pages in a first in first out (FIFO) order since “placeInArray” keeps track of the oldest element in the array.

LRU Implementation

Similarly to FIFO, initialize an integer variable named “placeInArray” to 0 before entering the while loop which will keep track of the current location in the cache array.

We will keep track of the least recently used page by keeping at the beginning of the array and having the most recently used page at the end of the cache array. Using a shifting algorithm, the ordering of the array represents the order in which it has been referenced.

Check if the requested page is already in the array. If it is, update the array by shifting all elements to the right of the found page one to the left and moving the requested page to the end of the array.

If the requested page was not found in the cache, increment the total number of faults. Then, check if there are any empty slots in the array with a for loop. If an empty slot is found, update “placeInArray” to that location and break the loop. Place the requested page at the current location.

However, if there are no empty slots, shift all of the elements of the cache array one to the left and insert the requested page at the end of the array. Thus removing the lru page.

Second Chance Implementation

To the structure of page (ref_page), add a boolean variable named “bit”. When initializing the elements of the cache array, set “bit” to 0 for all elements. Initialize the integer variable “placeInArray” to 0 to keep track of the current location in the cache array.

While receiving page requests, check if the requested page is already in the cache. If it is, set the “bit” to 1, signaling that it is being given a second chance and break out of the loop.

However, if the requested page was not found, increment the total number of page faults. Look for the first page with its “bit” set to 0. If a page’s “bit” is 1, set it to 0 and continue looking, this will prevent that page from being replaced this time around thus satisfying the second chance requirement. Once a page with “bit” 0 is found, place the new page at that location, then increment “placeInArray” appropriately.

Test and Results

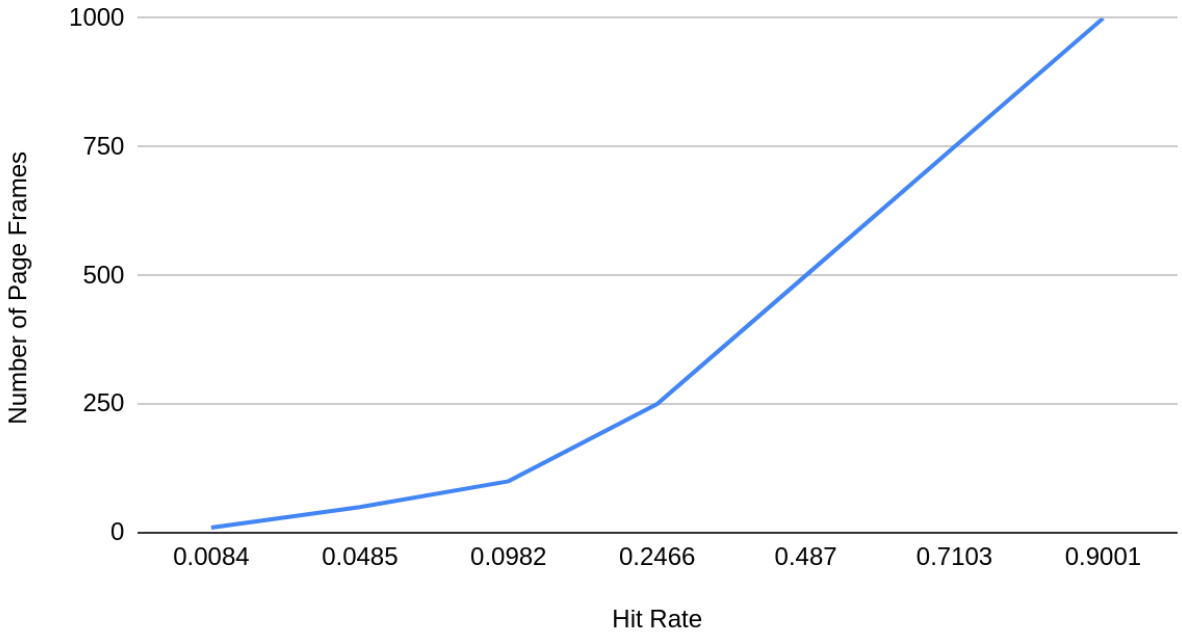
Tests with “testInput.txt”
Requests = 20
Cache Size = 10

Algorithm	Number of Page Faults	Hit Rate
FIFO	16	0.2
LRU	16	0.2
Second Chance	16	0.2

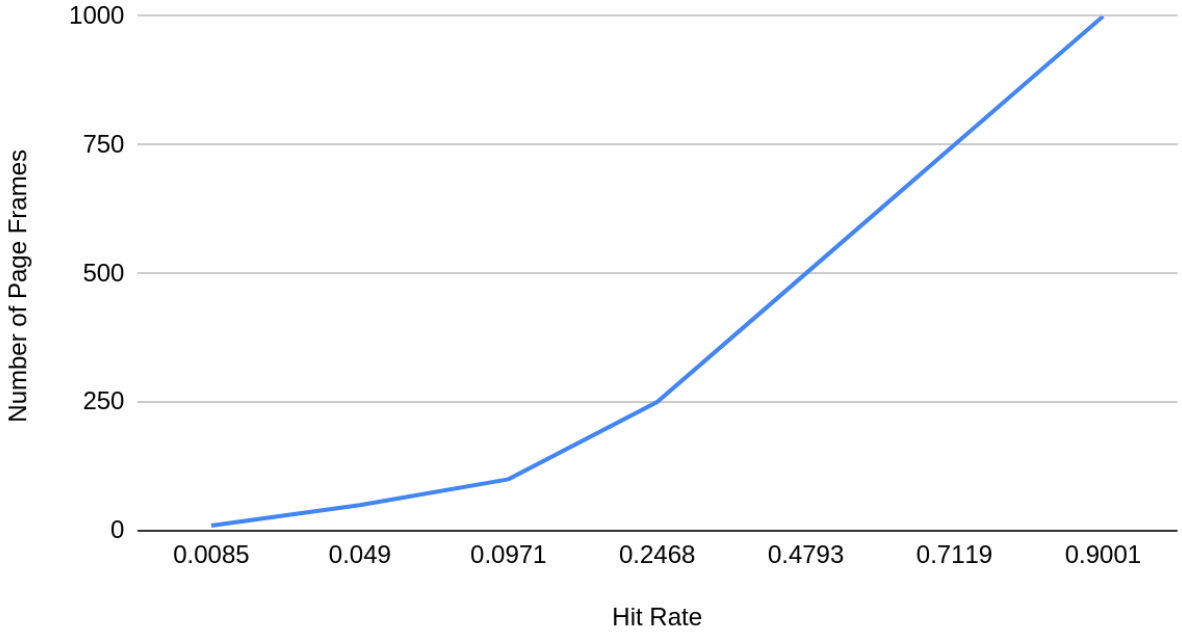
Tests with “accesses.txt”
Requests = 10,000

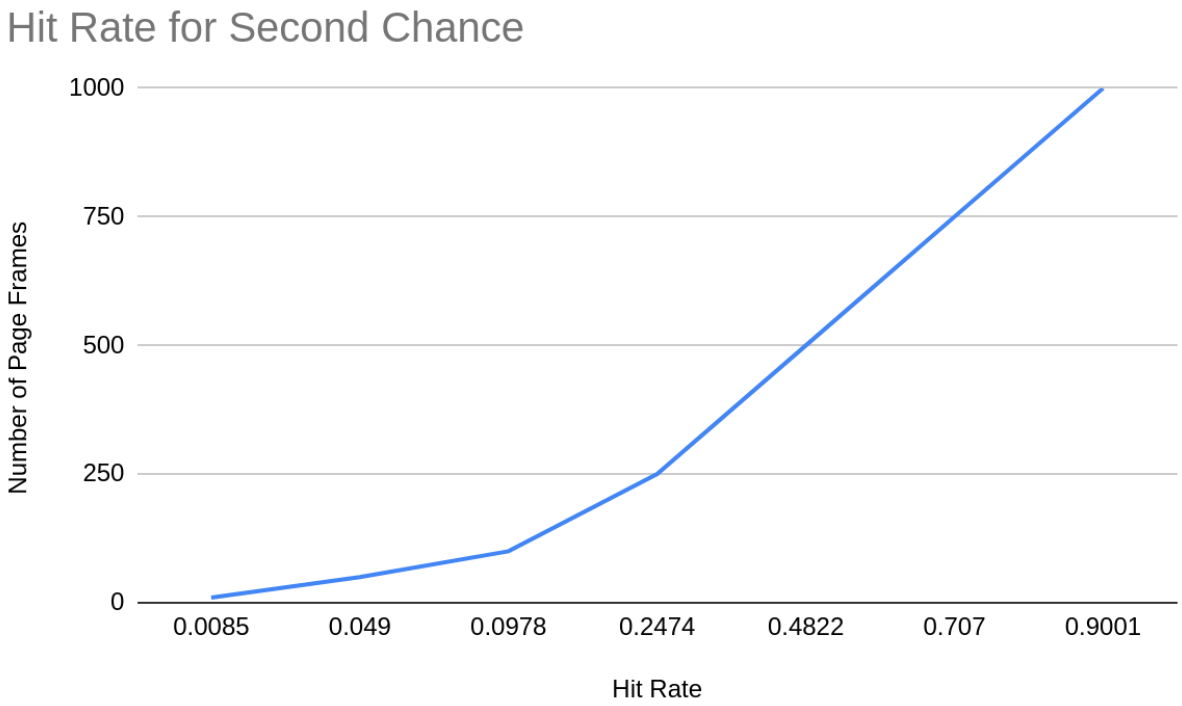
	FIFO		LRU		Second Chance	
# of frames	# page faults	Hit Rate	# page faults	Hit Rate	# page faults	Hit Rate
10	9916	0.0084	9915	0.0085	9915	0.0085
50	9515	0.0485	9510	0.049	9510	0.049
100	9018	0.0982	9029	0.0971	9022	0.0978
250	7534	0.2466	7532	0.2468	7526	0.2474
500	5130	0.487	5207	0.4793	5178	0.4822
750	2897	0.7103	2881	0.7119	2930	0.707
1000	999	0.9001	999	0.9001	999	0.9001

Hit Rate for FIFO



Hit Rate for LRU





For all algorithms, the hit rate improves the more page frames are allocated. The relation does not appear to be linear, in fact, it appears to follow a logarithmic trend. All three algorithms have very similar performances with little deviation in their results. Naturally, the hit rate will hit 1 consistently when the number of page frames is equal or greater to the number of requests.