Eva Stenberg
COEN 177L - Thursday 2:15
Lab 8 - Memory Management

Implementation of Page Replacement Algorithms

FIFO - FIFO was easily implemented using the given code. It works by checking if the requested page is in the cache, and if true, it sets the boolean variable to true and then breaks. If false, it prints the page of the page fault, replaces the current first page with the next, increments the page fault counter, and increments the counter variable to keep track of the next page in the queue.
Results using accesses.txt and page size of 50:

```
10000 Total Memory Requests
9515 Total Page Faults
Miss Rate = 0.951500
Hit Rate = 0.048500
```

LRU - This algorithm sets the current page of the outer for-loop to be the oldest item in the queue. If the page is found, page entries that are younger than the current oldest entry have their time variable incremented (because they haven't been used), the boolean variable is set to true, and the time variable for the found entry is set to 0 since it was just accessed. The page entries that haven't been used have time incremented. If the page is not found in the cache, then the oldest page in the queue is found, the page at which the fault occurred is printed, the oldest page in the queue is replaced with the new page and its time variable set to 0, and the fault counter is increased.
accesses.txt and page size of 50:

```
10000 Total Memory Requests
9510 Total Page Faults
Miss Rate = 0.951000
Hit Rate = 0.049000
```

Second chance - Second chance algorithm initially searches for the page request and if it is found in the cache, then it sets the flag variable to 1 which indicates it's already been read once and is on its second chance. If the page request is not found, the fault counter is increased and the algorithm looks for the first page entry that has its flag set to 0, meaning it's already had two chances, while keeping track of the last checked page in the queue. When the algorithm finds the next page with flag set to 0, it replaces this page with the new page, sets its flag to 0, and prints the page at which the fault occurred.

accesses.txt and page size of 50:

10000 Total Memory Requests
9510 Total Page Faults
Miss Rate = 0.951000
Hit Rate = 0.049000

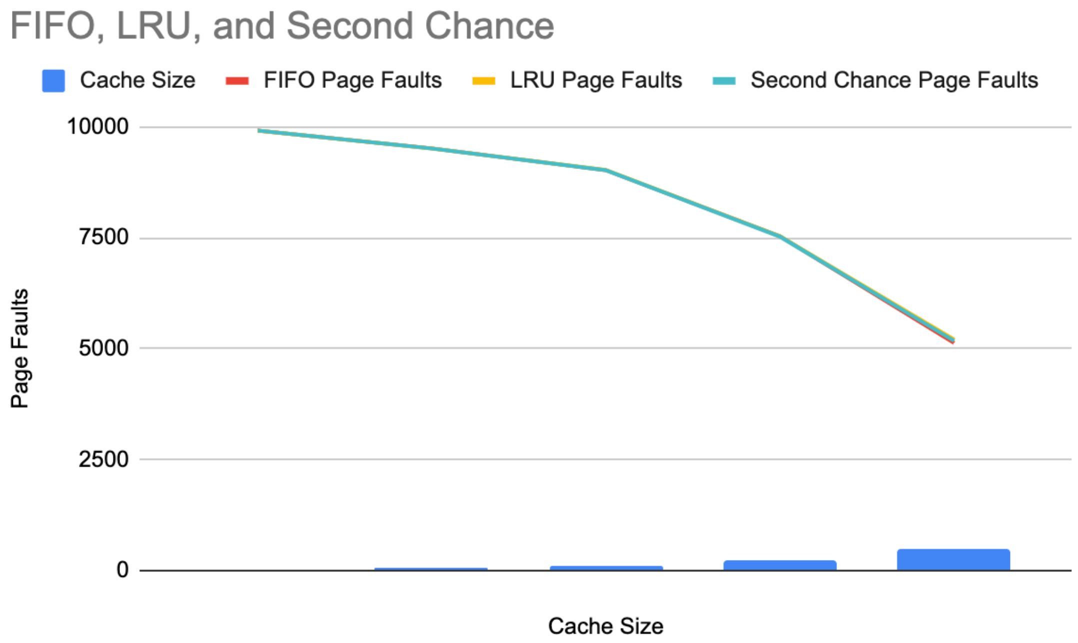
Step 6: Results

Cache Size	FIFO Page Faults	LRU Page Faults	Second Chance Page Faults
10	9900	9898	9897
50	9511	9514	9507
100	8970	8966	8962
250	7444	7436	7415
500	5084	5049	5052

Table 1: testInput10K.txt

Cache Size	FIFO Page Faults	LRU Page Faults	Second Chance Page Faults
10	9916	9915	9915
50	9515	9510	9510
100	9018	9029	9022
250	7534	7532	7526
500	5130	5206	5178

Table 2: accesses.txt



Judging by these results, the algorithms have a very similar performance in hit rate for the accesses.txt file. When the cache size was increased, there was an obvious change in page faults decreasing.