Hanna Albright

# Lab assignment 8: Memory Management

## Test Sample Page Faults

In this lab, I created an array to act as a cache to be implemented using FIFO, LRU, and the second chance page replacement algorithm. The lower the index for the page in the cache, the closer it is to being removed. I used a switch case so that you can determine which algorithm you want to use in the command line.

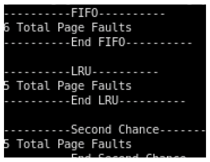
**./lab8 filename.txt cachesize algorithm**

filename.txt = name of plain text file with an integer on each line
cachesize = maximum number of pages (integers) held in the cache array
algorithm = integer (0 for FIFO, 1 for LRU, 2 for second chance)

I started by coding a version of LRU which extends into the FIFO switch case. The LRU case says if a page is found in the cache, then move it to the end of the array. The FIFO case checks if the page is in the cache, and if not, it places that page at the end of the array and removes first page. A break is coded between the FIFO  In the second chance case, if the page is not in the cache, it cycles through the array checking if the first page in the array has a reference bit of 1. If it doesn't, flip that page's reference bit to 1 and place it at the back of the cache. If the page has a reference bit of 1, remove it and place the new page at the back of the cache. In every algorithm, it keeps track of the number of page faults and then prints it to the terminal at the end.

To test my code, I created a file with 8 numbers to lookup using the three paging algorithms I coded. Since my test had such few page requests, I also used a small cache of size 2. After running each page request algorithm, I was able to calculate the miss-rate of ¾ for FIFO, ⅝ for LRU, and ⅝ for second chance.

**Step 6 output results**

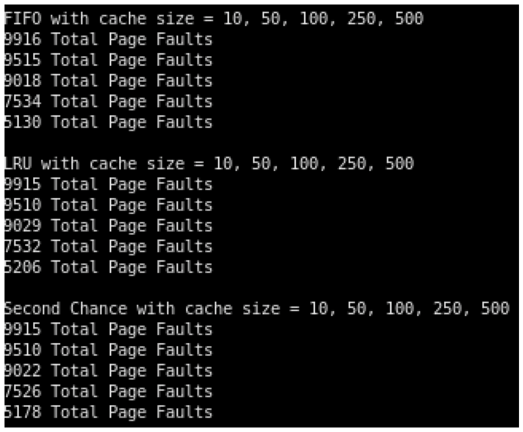


```
----------FIFO----------
6 Total Page Faults
----------End FIFO----------

----------LRU----------
5 Total Page Faults
----------End LRU----------

----------Second Chance-------
5 Total Page Faults
```

## Hit Rates for accesses.txt

The accesses.txt document has a list of 10,000 page requests, so you can get a more conclusive understanding of how cache size affects the miss rate, and therefore the hit rate. I created three graphs that show how much the hit rate increases when the cache increases. I used cache sizes of 10, 50, 100, 250, and 500 and used their page fault total to construct a graph for each algorithm. FIFO and LRU were very similar through any cache size while the second chance algorithm started to differentiate itself around a cache size of 250.

**Code Output for 10,000 Requests**



```
FIFO with cache size = 10, 50, 100, 250, 500
9916 Total Page Faults
9515 Total Page Faults
9018 Total Page Faults
7534 Total Page Faults
5130 Total Page Faults

LRU with cache size = 10, 50, 100, 250, 500
9915 Total Page Faults
9510 Total Page Faults
9029 Total Page Faults
7532 Total Page Faults
5206 Total Page Faults

Second Chance with cache size = 10, 50, 100, 250, 500
9915 Total Page Faults
9510 Total Page Faults
9022 Total Page Faults
7526 Total Page Faults
5178 Total Page Faults
```

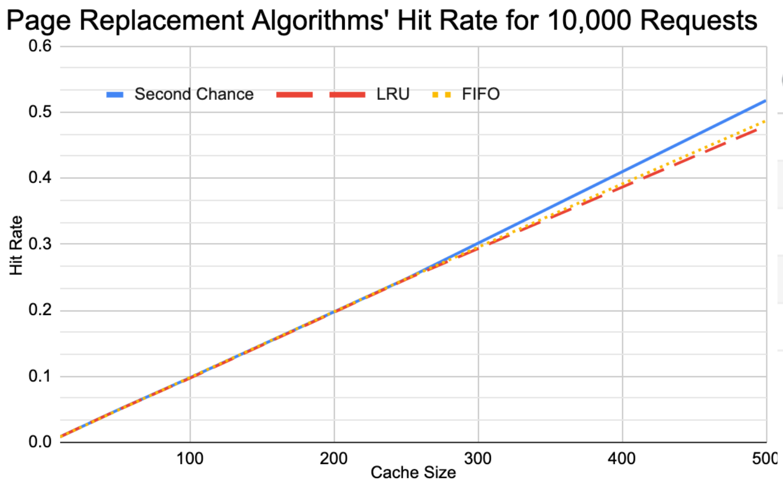
Hanna Albright

I was able to generate this data by using a shell script consisting of each terminal command I would have to run. Each command in the shell script file used this pattern:

```
cat pagerequestfile.txt | ./lab8 cachesize algorithm
```

pagerequestfile.txt = name of plain text file with an integer on each line
cachesize = maximum number of pages (integers) held in the cache array
algorithm = integer (0 for FIFO, 1 for LRU, 2 for second chance)

**Hit Rate Graph for 10,000 Requests**



| Cache Size ▲ | FIFO | LRU | Second Chance |
|---|---|---|---|
| 10 | 0.0084 | 0.0085 | 0.0085 |
| 50 | 0.0485 | 0.049 | 0.049 |
| 100 | 0.0982 | 0.0971 | 0.0978 |
| 250 | 0.2476 | 0.2468 | 0.2474 |
| 500 | 0.487 | 0.4794 | 0.5178 |

The hit rate for the FIFO algorithm stays linear  with an increasing cache size except for a slight decrease starting after a cache size of 200. The LRU algorithm's data is extremely similar to the FIFO graph as the hit rate starts decreasing at a cache size of 200 and even more after a size of 400. The similarity between the FIFO and LRU data makes sense since their algorithms are extremely similar, which I used to my advantage while programming. I constructed the FIFO algorithm so that if the page being requested isn't found, it is placed in the back of the array after the first page, which is the first in, is removed. If the cache isn't full and it isn't already there, the page is placed after the last page added in the array. I placed the case for LRU about the FIFO case and didn't place a break inbetween so that the LRU code includes the FIFO code. The LRU case makes it so that if the page is found in the cache, it gets placed at the end of the array and acts as resetting the timer.

The second chance algorithm, if the page isn't found in the cache and the cache is full, check the first page in the array has a reference bit of 1. While it's 1, switch that page's reference bit to 0 and place it in the back of the array after shifting all elements to the left by one. Once a page with a reference bit of 0 is found, remove that page from the array and replace it with the new page whos reference bit is 0. If the page is found, make sure it's reference bit is 1 so that it can be given a second chance.

In conclusion, the three algorithms' data were very similar until after the cache size increased to about 250, when the second chance's hit race passed LRU and was almost the same as FIFO.