Final Lab Assignment

TASK:

The essential question to this project will be: How well do different page replacement algorithms perform given different circumstances? The purpose of this project will be to determine which algorithms best suit which specific circumstances, if any at all. The project will also aim to find the most ideal algorithm based off of time per workload in practice by timing runs. The “best” algorithm will also be judged based off of its simplicity of implementation and obviously its effectiveness to limit page faults. By the three criteria, I will give each algorithm a score based off of hard evidence or by subjectivity if necessary (for example, the simplicity criteria). The page replacement algorithms that will be tested will be: Second Chance, LRU, FIFO, NFR, NRU, Clock, and Random.

Since there are 7 different algorithms, I will rank give each algorithm a score from 1(best)-7(worst). The data that will be presented at the conclusion of the research will include: the results of rankings per criteria, the average of all timing runs, and any specific timing run where the page request inputs show favor to a specific page replacement algorithm. I will show the outliers of specific page inputs because the goal of the project is not to find the best algorithm, but rather to find if/how certain workloads favor a specific algorithm.

PROJECT DETAILS:

We will use three different sets of workload data, in addition to varying the cache sizes for each algorithm. The first workload will be the data provided to us from the previous lab. The workloads will be a randomly generated set of pages with shorter and longer strings of digits. The cache sizes will vary from a size of 50, 100, and 200 pages.

The computer components used in this program include the following: Intel i7-8700 Processor, 32GB RAM, with the use of Ubuntu on Virtual Machine by Oracle. The reason that a virtual machine was used is because “getline” would not operate on Windows, and the Cygwin terminal was not cooperating with me during the duration of the project.

In this project, Clock was not cooperating well with me, so all data points have been copied from Second Chance (SC) due to their similarities of execution.

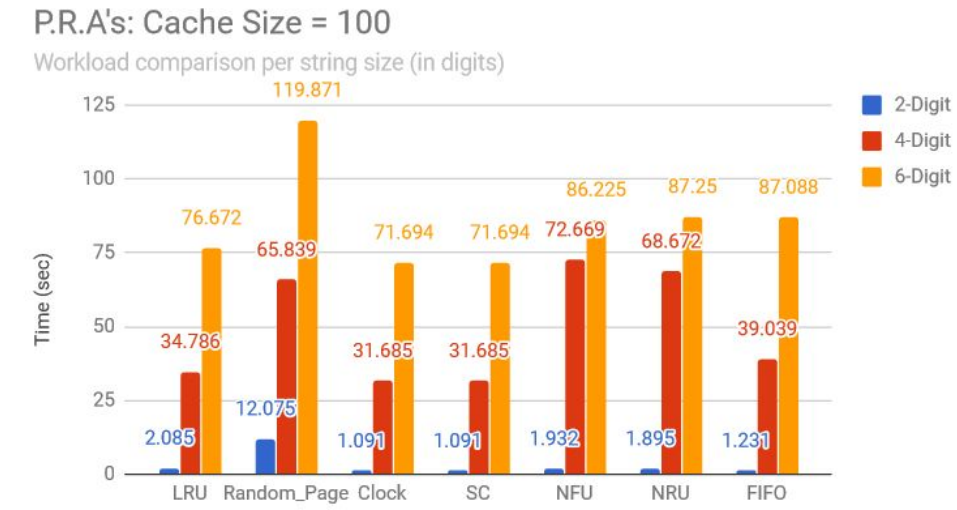
FILES USED AND COMPILING INSTRUCTIONS:

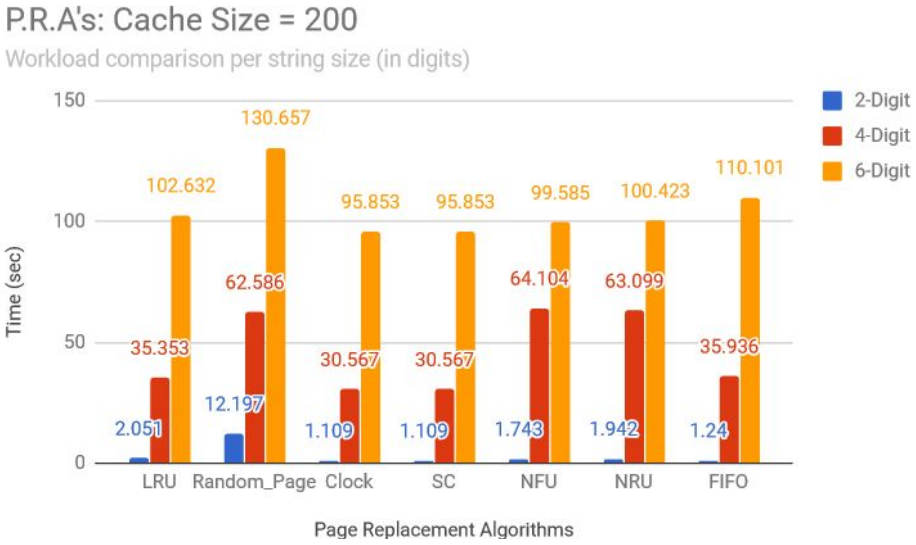
The C files used include sc.c, lru.c, fifo.c,clock.c, random.c,nfu.c, and nru.c. Additionally, the data will come from digit_2.txt, digit_4.txt, and digit_6.txt. To compile the files, please first pipe in the .txt file into the desired .o file, then provide the cache size.
For example:

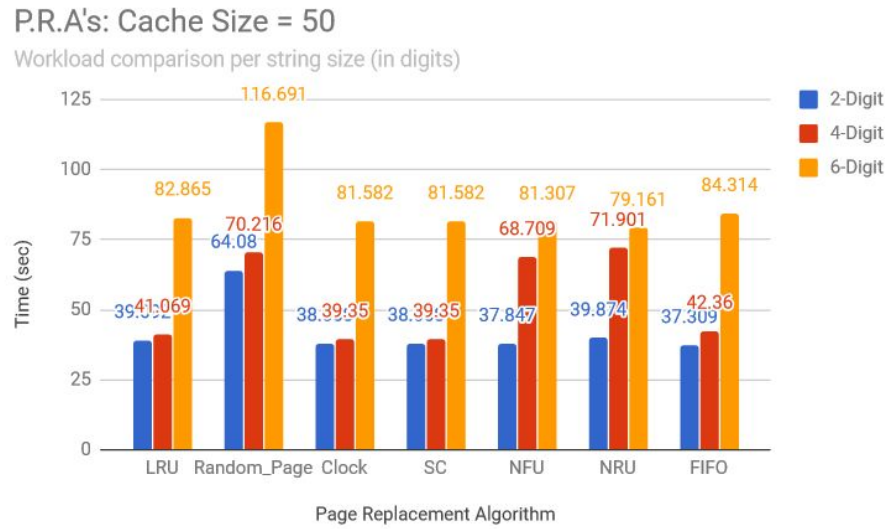
```
$ gcc -o lru.o lru.c
$ cat digit_6.txt | ./lru.o 100
```

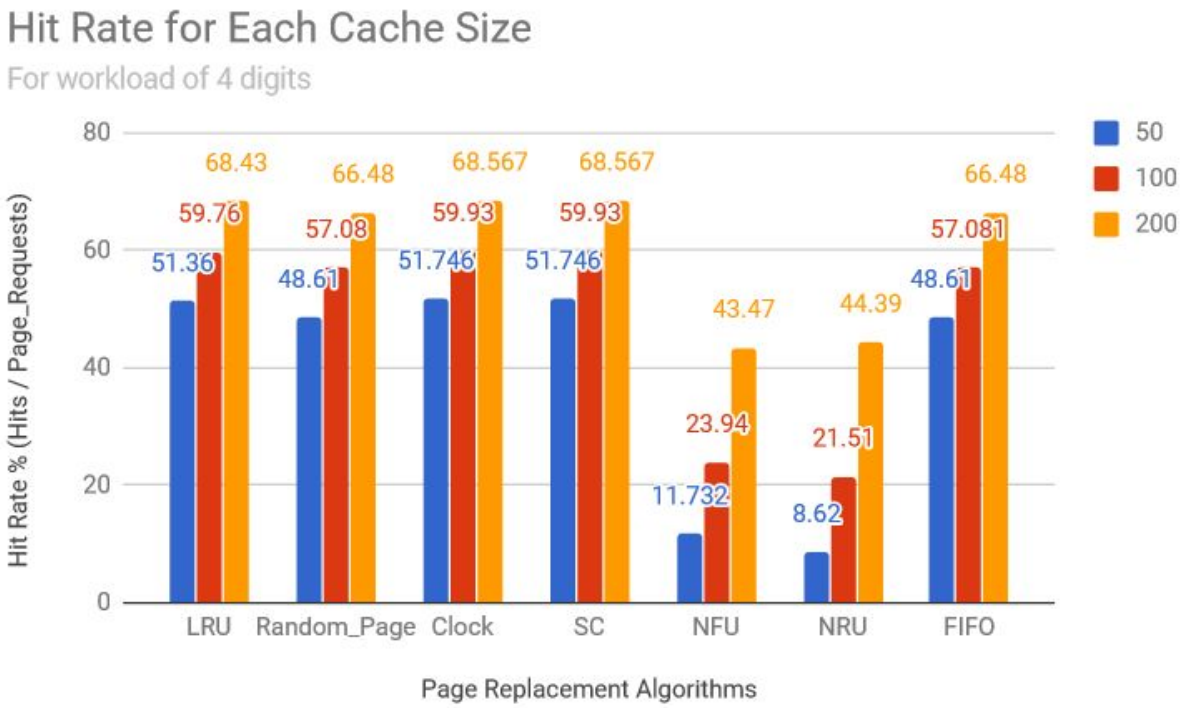
For more specific details about each file, please read the comments within the desired file, which will provide the details of how it works, my personal choices of structure, and the input/output for relevant functions.

RESULTS:


P.R.A's: Cache Size = 100
Workload comparison per string size (in digits)
Time (sec)
125
100
75
50
25
0
2-Digit
4-Digit
6-Digit
2.085
34.786
76.672
12.075
65.839
119.871
1.091
31.685
71.694
1.091
31.685
71.694
1.932
72.669
86.225
1.895
68.672
87.25
1.231
39.039
87.088
LRU
Random_Page
Clock
SC
NFU
NRU
FIFO


P.R.A's: Cache Size = 200
Workload comparison per string size (in digits)
Time (sec)
150
100
50
0
2-Digit
4-Digit
6-Digit
2.051
35.353
102.632
12.197
62.586
130.657
1.109
30.567
95.853
1.109
30.567
95.853
1.743
64.104
99.585
1.942
63.099
100.423
1.24
35.936
110.101
LRU
Random_Page
Clock
SC
NFU
NRU
FIFO
Page Replacement Algorithms


P.R.A's: Cache Size = 50
Workload comparison per string size (in digits)
Time (sec)
125
100
75
50
25
0
2-Digit
4-Digit
6-Digit
41.069
82.865
64.08
70.216
116.691
38.39
81.582
38.39
81.582
37.847
68.709
81.307
39.874
71.901
79.161
37.309
42.36
84.314
LRU
Random_Page
Clock
SC
NFU
NRU
FIFO
Page Replacement Algorithm



SIMPLICITY:

FIFO(1), Random(2), and LRU(3) are the outliers; they were by far the easiest to implement of the group. Next up would be NRU(4), and NFU(5), followed by Second Chance and Clock(t6). These rankings are completely subjective as previously mentioned and were ranked based off the length and complexity of each algorithm. While each used a linked list, some involved more care with the pointers and placement of each object. Ultimately this category means very little in terms of performance, but is a measure of convenience for the programmer.

HIT RATE:

The graph does not include the other workloads because every algorithm scored nearly 100% hit rate for a 2 digit workload and had nearly 0% for a 6 digit workload. It is clear from the graph that NRU(7), NFU(6), FIFO(5), and Random(4) were the worst performers in hit rate percentage. When looking at the averages of each SC and Clock(t1) were the best, followed by a close LRU(3). My main takeaway is that the Random algorithm outperformed two other algorithms by astonishing numbers, and another by 0.001% (FIFO). All in all, SC/Clock and LRU are the best for the hit rate, and are almost the exact same between all three.

EXECUTION TIME:

It is clear in all instances that Random(7) and FIFO(6) are the worst algorithms for each cache size and workload. A tight range between NRU(5), LRU(4), and NFU(3). LRU found its way in between NFU and NRU because of its poor performance in a cache size other than 100.

This will leave SC and Clock(t1) as the leaders for all cache sizes and workloads. The quicker the better, especially for consistency across each cache size. For consistency, SC/Clock takes first because of LRU's inconsistency

When averages out, my scores record: tied first: SC/Clock(2.66), third:LRU(3.33), fourth: FIFO(4.0), fifth: Random(4.33), sixth: NFU(4.66), seventh:NRU(5.33). However, LRU has managed to stay in the top 4 for each category, making it the most consistent of the algorithms. In summary, SC/Clock wins the average for my ranking of best algorithm, but LRU is notable for its strong consistency of high performance and close followings to SC/Clock in the major performance categories.

To answer the essential question, LRU is the most consistent in high-performance, but is very inconsistent when it comes to cache size. Cache size seems to be the most dominant factor in breaking apart algorithms in terms of time, as it is for FIFO. For hit rate, every algorithm has extremely similar results for each cache size and workload. The smaller and larger workloads were not included for in the graph for simplicity's sake, and besides NFU and NRU, each algorithm is maxly differenced by 3.1%.