CSE 330 – Project 3 – Full Deadline

# Design

I developed an ARC cache in C++. It is contained in ARC.cpp and ARC.h. Unfortunately, it does not seem to work. I compared it to your given algorithm, and it has nearly a one-to-one correspondence.

I could not get results from my caching program because it does not seem to work.

CSE 330 – Project 3 – Milestone

# 1. LRU Replacement Policy Implementation

I used a hash map and a doubly-linked list to implement the milestone. The data type used for both was a size\_t (unsigned long int). The size\_t corresponds with the block number. The only floating point variables that are used are for calculating the hit ratio.

# 2. Table of Hit Ratios

Below is a table of the hit ratio (%) of each test file given, along with the LRU size it was tested on. I got the same results as your for the published test case results.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LRU Size | OLTP.lis | OLTP1000(1).lis | OLTP2000(1).lis | P12.lis | P3.lis | P31000(1).lis | P32000(1).lis | P4.lis | P6.lis |
| 1 | 0.008533 | 0 | 0 | 0.000197 | 0.00023 | 0 | 0 | 0.000147 | 0.001981 |
| 2 | 0.023082 | 0 | 0 | 0.001635 | 0.00046 | 0 | 0 | 0.000177 | 0.002052 |
| 4 | 0.042116 | 0 | 0 | 0.00296 | 0.000971 | 0 | 0 | 0.000566 | 0.003291 |
| 8 | 0.207188 | 0 | 0.1 | 0.03138 | 0.027298 | 0 | 0 | 0.016697 | 0.034738 |
| 16 | 0.840786 | 0.4 | 0.65 | 0.093384 | 0.075197 | 0.020088 | 0.010497 | 0.051107 | 0.066311 |
| 32 | 2.241111 | 1.1 | 1.75 | 0.206618 | 0.140988 | 0.050221 | 0.026242 | 0.096308 | 0.08802 |
| 64 | 5.259888 | 4.2 | 5.85 | 0.414227 | 0.212172 | 0.140619 | 0.073477 | 0.18623 | 0.106778 |
| 128 | 10.34366 | 8 | 10.05 | 1.022997 | 0.389301 | 0.405116 | 0.222179 | 0.412589 | 0.140332 |
| 256 | 16.69309 | 12.9 | 15.6 | 2.514299 | 0.600705 | 1.191911 | 0.738266 | 0.835024 | 0.268455 |
| 512 | 23.69263 | 15.9 | 20.35 | 3.4672 | 0.816353 | 2.678452 | 1.770438 | 1.914499 | 0.428484 |
| 1024 | 33.21847 | 16.4 | 28.85 | 4.084843 | 1.051707 | 3.284452 | 2.263781 | 2.685061 | 0.708153 |
| 1536 | 39.01777 | 16.4 | 29.3 | 4.48059 | 1.11622 | 4.529932 | 3.100016 | 2.833442 | 0.790815 |
| 2048 | 42.77735 | 16.4 | 29.3 | 4.831232 | 1.153972 | 4.590197 | 3.154248 | 2.963857 | 0.859343 |
| 3072 | 47.38811 | 16.4 | 29.3 | 5.294184 | 1.242767 | 4.824561 | 3.336191 | 3.185595 | 0.941105 |
| 4096 | 51.24045 | 16.4 | 29.3 | 5.611635 | 1.321235 | 4.84465 | 3.518133 | 3.315959 | 1.093613 |
| 6192 | 56.01081 | 16.4 | 29.3 | 6.002318 | 1.472344 | 4.928351 | 3.958993 | 3.521546 | 1.188238 |
| 8192 | 58.86113 | 16.4 | 29.3 | 6.221231 | 1.622764 | 4.928351 | 3.958993 | 3.650363 | 1.26405 |
| 12288 | 62.69235 | 16.4 | 29.3 | 6.670094 | 1.842782 | 4.928351 | 4.177673 | 3.845644 | 1.461073 |
| 16384 | 65.42464 | 16.4 | 29.3 | 7.091846 | 2.07627 | 4.928351 | 4.177673 | 4.073833 | 1.693268 |
| 24576 | 68.90669 | 16.4 | 29.3 | 7.884136 | 2.578417 | 4.928351 | 4.186421 | 4.489163 | 2.57039 |
| 32768 | 71.19625 | 16.4 | 29.3 | 8.926302 | 3.567657 | 4.928351 | 4.196918 | 5.239372 | 4.240371 |
| 49152 | 73.55463 | 16.4 | 29.3 | 11.50108 | 6.662969 | 4.928351 | 4.196918 | 8.919417 | 9.319567 |
| 65536 | 75.87177 | 16.4 | 29.3 | 14.43096 | 12.71992 | 4.928351 | 4.196918 | 10.75842 | 17.76133 |

Below is a graph, showing how the LRU cache size affects the hit ratio.

# Status

The LRU replacement algorithm is working perfectly and efficiently.

# Interactions

I did not interact with anyone to solve this assignment.

# Citations

None – didn’t even look at books or language documentation.