FINAL PROJECT REPORT

GROUP 13

Mandeep Bawa, K. Sai Anuroop, Ashish Kumar Bhoi

17th November 2019

Problem Statement

To provide page buffering for the PF layer using two page-replacement strategies to be chosen for a file at the time of opening it: LRU and MRU.

Review of PF Layer

We figured out the workflow of PF Layer and understood various dependencies among different files in this layer.

Code

For implementing the project, we chiefly edited the files buf.c, pf.c, pf.h, pftypes.h and main.c. We have obtained the following statistics by defining the following variables and calculating the values as defined below.

Variables implemented:

1. Number of buffer read requests

   buf_hit + buf_miss

   Calculated as the sum of buffer hits and buffer misses.

2. Buffer hit rate

   int buf_hit

   Calculated as the ratio of buffer hits and buffer read requests.

3. Buffer miss rate

   int buf_miss

   Calculated as the ratio of buffer misses and buffer read requests.

4. Number of buffer evicts

int num_buf_evicts

A counter is incremented every time a page is evicted from the buffer.

5. Number of logical I/Os

   int logical_ios

   Calculated as the read requests which are catered to by the buffer alone.

6. Number of physical I/Os

   int physical_ios

   Calculated as the sum of buffer misses and buffer flushes.

7. Average lifespan of an entry in buffer

   int avg_lifespan

   Calculated as the average number of read requests made to the buffer between two consecutive buffer evicts.

Functions implemented in *buf.c*:

1. *Buf_getstats()*
   Prints the above statistics after a test file executes.

2. *fetch_victim_lru()*
   Fetches the page based on LRU replacement policy.

3. *fetch_victim_mru()*
   Fetches the page based on LRU replacement policy.

We have also written a shell script for executing the test files.

Working Method

We varied the values of PF_MAX_BUFS (buffer size) as 20, 40 and 60, gauging the buffer's performance for different replacement policies of LRU and MRU for the three test files provided. We then plotted graphs for the obtained statistics.

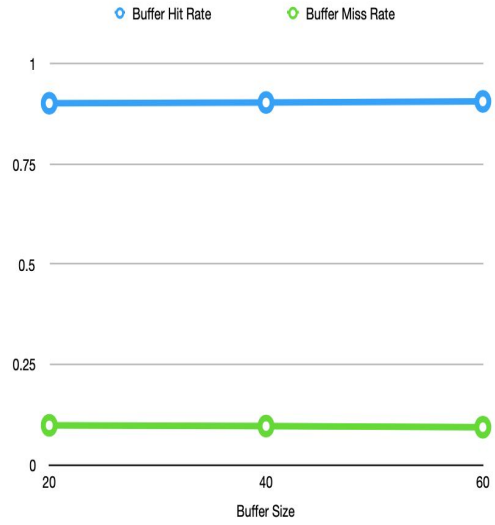
Results Obtained

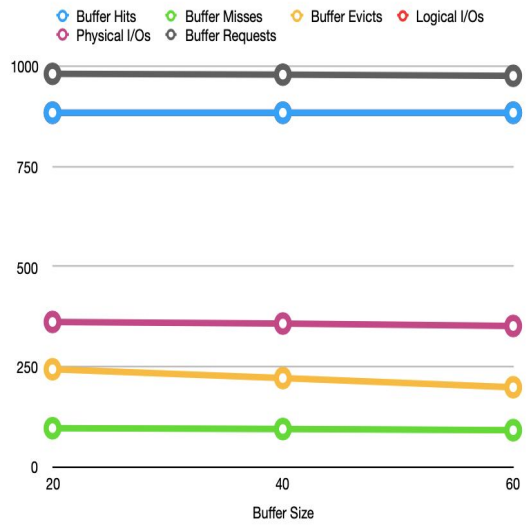
test2.c LRU

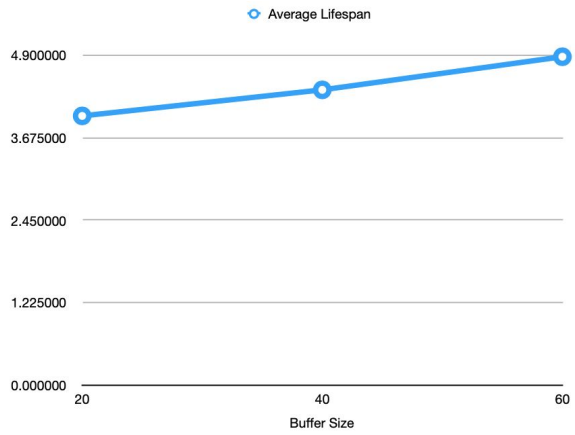
Buffer Size	Buffer Hits	Buffer Hit Rate	Buffer Misses	Buffer Miss Rate	Buffer Evicts	Logical I/Os	Physical I/Os	Average Lifespan	Buffer Requests
20	884	0.901121	97	0.098879	244	884	362	4.000000	981
40	884	0.902962	95	0.097038	222	884	358	4.387387	979
60	884	0.905738	92	0.094262	199	884	352	4.879397	976

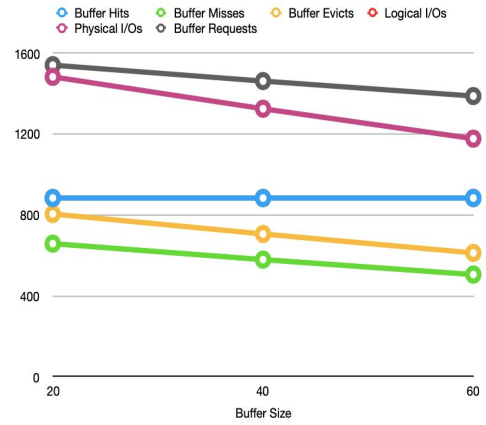
test2.c MRU

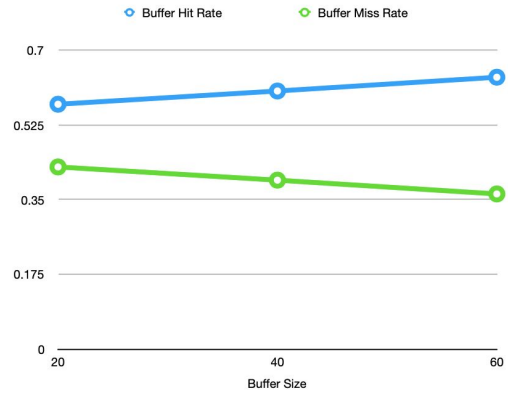
Buffer Size	Buffer Hits	Buffer Hit Rate	Buffer Misses	Buffer Miss Rate	Buffer Evicts	Logical I/Os	Physical I/Os	Average Lifespan	Buffer Requests
20	884	0.573281	658	0.426719	805	884	1484	1.909317	1542
40	884	0.604238	579	0.395762	706	884	1326	2.065156	1463
60	884	0.636429	505	0.363571	612	884	1178	2.261438	1389

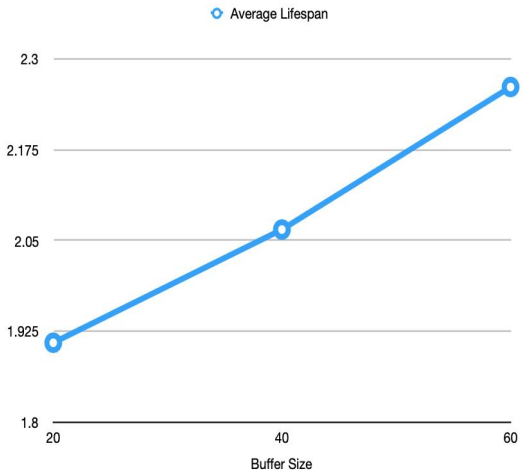












Observations and Analysis

The results obtained for test1.c and test3.c did not show any interesting variation in the statistics obtained, as there were very few pages being accessed by these files. Rather, it was for test2.c where considerable variation was observed, as described below.

Average Lifespan:
LRU (4 - 4.8) has more than MRU (1.9 - 2.2)
Buffer Hit Rate:
LRU (0.9) strategy has more hits than MRU (0.57 - 0.63)
Buffer Miss Rate:
LRU (0.09 - 0.1) has less misses than MRU (0.35 - 0.45)
Logical I/Os:
LRU and MRU perform equally in terms of logical I/Os

Physical I/Os:
LRU (350- 360) has less access to disk than MRU (1100 - 1400)

Conclusions

Since temporal locality is generally observed in the pages being accessed, LRU outperforms MRU by a considerable margin.

Appendix

Github Link for code