IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Performance Improvement Of LRU Page Replacement Algorithm

[1] Greetta Pinheiro, [2]Madhubala Rathore

[1]Research Scholar, [2]Assistant Professor

[1]SC&SS Jawaharlal Nehru University, New Delhi,

[1]Government College Chourai, Chhindwara, Madhya Pradesh, India

Abstract: The performance of the page replacement algorithms can be improved by augmenting it with some data along with the algorithm design. This can indeed optimize the time taken for the swapping process. In this system we are augmenting two data structures with LRU page replacement algorithm, they are doubly circular linked list and hash table. Doubly circular linked list is used because the traversal between the nodes is easier and faster. Hashing is more efficient in performing optimal searches and retrievals because it increases speed, better ease of transfer, better retrieval and reduce overhead. Furthermore Hashing can reduce disk space and access time by inserting and retrieving a page from the main memory in only one seek. Further we are comparing the performance of the classical LRU page replacement algorithm with the augmented LRU page replacement algorithm in terms of runtime.

Index Terms **- Least Recently Used Page Replacement Algorithm, Optimal page replacement Algorithm, Main Memory, Cache Memory, Doubly Circular Linked List, Splay Trees, Hash Tables, Skip list.**

I. INTRODUCTION

Computer system employs caching techniques to speed up the access to pages [8]. Various page replacement algorithms are used for swapping in and swapping out the pages from the cache memory. Whenever an application requests access to a page(s) on disk, the operating system retrieves the requested page(s) or the portion of the requested page and stores it in the cache memory. Subsequent accesses to these pages which are already accessed and are residing inside the cache memory can be performed by accessing it from the cache rather than accessing it from the disk. Accessing the cache is much faster than accessing the disk, performance is improved especially when the page is frequently accessed.

One among the common and widely used page replacement algorithm is LRU (Least Recently Used) [1][2]. The LRU page replacement algorithm choose a page that has not been accessed for the longest time for swapping out of the cache memory and swaps in the requested new page in place of the swapped out page.

For example, suppose the cache memory can store three pages, page 1 was accessed 10 seconds ago, page 2 was accessed 20 seconds ago, and page 3 was accessed 30 seconds ago. Thus, when a new page is to be cached, page 3 will be discarded to make room for the new page to be cached because page 3 was least recently accessed [8]. i.e. it was residing in the cache memory without getting accessed for longest time.

LRU is hard to implement but its performance is fairly close to the Optimal Page Replacement algorithm, which is taken as a benchmark for the comparison of various page replacement policies. Furthermore it has been proved that LRU can never have a result more than N-times more page faults than Optimal Page replacement algorithm, where N is proportional to the number of pages in the managed pool. Main advantage of the LRU algorithm is that it is amenable to full statistical analysis. Hence the optimization of LRU page replacement policy can indeed increase the performance of the Cache memory.

There are three main operations which are carried out every page replacement algorithm: inserting a page into the cache, deleting a page from the cache and searching a page. All these operations are dependent on various data structures that are been employed in the form of page table. The scheme and complexity of each these data structures vary. Therefore, an efficient data structure or combination of various data structures can indeed improve the performance of page replacement algorithm.

The usage of the data structure in its original form might not serve in improving the efficiency of the page replacement algorithm. In order to make the data structure efficient for our purpose, we have to do some modification and use them. Modifying the data structures by adding additional parameters or information is called data augmentation. It is not always possible that data augmentation can give better performance, it needs to be verified by making use of the trial and error method [1][5].

In our proposed system we use two different data structures namely doubly circular link list and hash table for augmentation to LRU page replacement algorithm. Improving the performance is the main goal behind the design of any page replacement algorithm [2]. Our goal is also to improve the performance of LRU page replacement algorithm in terms of runtime using these augmented data structure[4].

II. LITERATURE RE REVIEW

The design of page replacement algorithms were mainly focused on the computational time. Efforts were made to reduce the computation time but the I/O time was not considered. The access time of main memory and external memory is different. The main memory access time is measured in terms of Nano seconds, whereas the access time of external memory such as disk is in terms of milli seconds, here there is a large difference between the access time of both these memories [5]. Whenever a page is accessed from the main memory, it is called as a hit and it takes less access time. If the page is not there in main memory, this can be termed as a miss. Here the page should be accessed from the disk and it takes more access time.

The access time can be reduced if we can increase the number of hits or the hit ratio. This can increase the efficiency of large computations where the amount of data required for the application will be large than the main memory size, for example computational biology, database systems, VLSI verifications, image processing etc.

The main idea behind LRU page replacement algorithm is that, the pages which are referred heavily in the last few instructions can have a chance to be referred in the next few instructions as well. So those pages should be retained in the main memory. When a page fault occurs, the LRU algorithm replaces the page which is not been used for the longest time.

The hybrid of LRU and LFU performs better than other page replacement algorithms such as Second-chance Frequency Least Recently Used (SF-LRU), Least Recently Frequently Used (LRFU), Weighting Replacement Policy (WRP) etc. [9],[11],[10].

In Linux systems, LRU page replacement along with tree which has three levels of page table is been used. These three levels include page directory, page middle directory and page table entries [5].

In our proposed system we implement LRU page replacement algorithm with augmented doubly circular link list and hash table in order to improve the performance.

III. DETAILED PROBLEM DEFINITION

Main memory is the storage element which is used to store the programs and data. Whenever the processor requires accessing data, it is copied from secondary memory to main memory [3]. Retrieving the data from the main memory, takes a considerable amount of time for the processor. In order to reduce it, cache memory is been employed. Access time for the cache memory is less than the main memory thus speeding up the access to pages [7]. When a processor requires a page, it initially checks the cache memory. If the page is found (i.e a hit), it will be accessed and used for further processing. Thus main memory is not used for accessing the page if the page is found in cache. Whereas if the page is not found in cache (page fault or miss), main memory will be accessed. The required page will be swapped out from the main memory and swapped in to the cache at some location. Various page replacement algorithms are used for swapping in and swapping out pages from cache memory. One among the widely used page replacement algorithm is LRU. It chooses a page which has not been accessed recently and swaps in the new requested page in place of it. The time for searching a page in the cache memory is not constant. It varies from one data structure to another. An LRU implementation using stack may take longer time than that implemented using a queue. The main goal of any page replacement algorithm is to improve the performance in terms of runtime. This access time can be reduced by augmenting some data structures. Our proposed system uses augmentation of doubly circular linked list and hash table for designing the LRU page replacement algorithm which in turn reduce the runtime.

IV. SOLUTION METHODOLOGY

A. Augmentation of Doubly Circular link list

The technique used for designing the doubly circular linked list for LRU page replacement algorithm is by making use of moved to Front (MTF) self-organizing heuristic technique [6]. The heuristic that moves the referred page to the head of the doubly circular linked list (Refer Figure 1) so that it is found out faster next time. This technique can speed up the linear searching performance if the referred page is likely to be searched again. This technique design improves the efficiency of the linear search of the referred pages by moving the more frequently accessed page towards the head of the doubly circular linked list (Refer Figure 2).

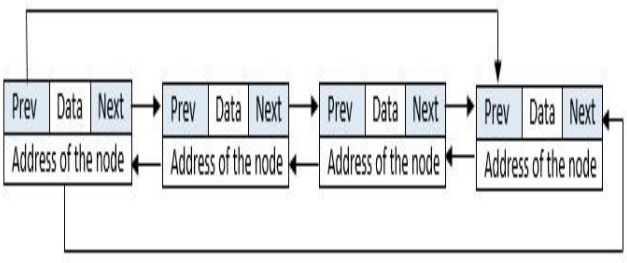

Figure 1. Doubly circular linked list

B. Augmentation of Hash Table

Hash table data structure makes use of a hash function to map the keys with the array positions. Out of the many different hashing techniques available we are using chaining technique of hashing so that unlimited number of collisions can be handled and also unlimited number of elements can be stored. Here we don't require apriori knowledge of the number of elements required to be stored.

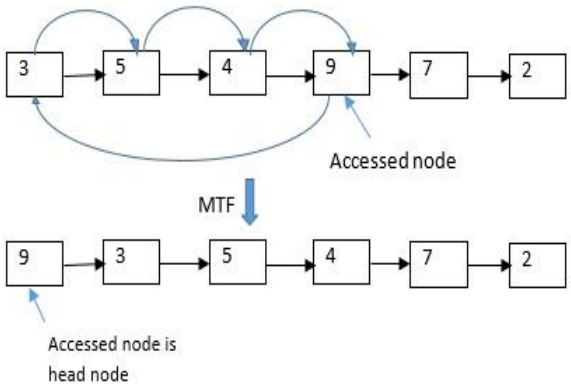

Figure 2. MTF Technique

V. Details of Analysis

In a linked list if a particular page is to be searched, each page (node) in the list has to be sequentially compared with the referenced page till the required page (node) is reached. Thus if we have to retrieve the n^{th} page it takes O(n) operations.

In contrast the doubly circular linked list rearranges the pages keeping the most recently accessed page at the head of the list. Generally in this searching technique the chance of accessing the page which has been accessed many times before is higher than the chances of accessing the page that has not been accessed frequently. This keeps the most recently accessed page as the head of the list resulting in reducing the number of comparisons which is required on an average for reaching the desired page. This leads to better efficiency reducing the search query time.

Hashing is the technique which uses almost constant time in case of searching, insertion and deletion of an element. It is similar to an array with its index behaving as a key. Each of the key (index) can be used for accessing the page in a constant time. Hash table is a data structure which is used to access the value or a page using a hash function which generates a key corresponding to the associated page. Hashing is a more reliable and flexible method for page retrieval than any other table data structure in terms of speed. A significant amount of difference in retrieval time can been seen when the number of entries in the hash table is large that is in terms of thousands and more. It takes O(1) time for searching an item inside the hash table.

VI. Results and Discursion

The hardware specification of the system used for execution is OS X EI Capitan version: 10.11, processor: 1.4 GHz Intel core i5, Memory: 4GB 1600 MHz, Graphics: Intel HD Graphics 5000 1536 MB. The execution results obtained after running same parameter 10 times, we took the average value for each corresponding parameter. Thus we have tabulated the results as shown below,

Least Recently Used Page Replacement Algorithm (Using stack)								
Number of Frames in cache	3	4	5	6	7	8	9	10
Number of Page Faults	10	9	9	8	7	7	7	7
Number of Hits	2	3	3	4	5	5	5	5
Hit Ratio (%)	16.66	25	25	25	41.66	41.66	41.66	41.66
Runtime (ms)	0.016000	0.015000	0.016000	0.016000	0.015000	0.016000	0.016000	0.016000

Figure 3. Results of LRU Algorithm using stack

Least Recently Used Page Replacement Algorithm (Proposed)								
Number of Frames in cache	3	4	5	6	7	8	9	10
Number of Page Faults	10	9	9	8	7	7	7	7
Number of Hits	2	3	3	4	5	5	5	5
Hit Ratio (%)	16.66	25	25	33	41.66	41.66	41.66	41.66
Runtime (ms)	0.000027	0.000069	0.000065	0.000068	0.000068	0.000067	0.000067	0.000067

Figure 4. Results of LRU Algorithm using Doubly Circular Linked List and Hashing

Here we have compared the result of existing LRU page replacement algorithm using stack with our proposed augmented LRU page replacement algorithm. The above results shows that while using the same dataset on both the algorithms, proposed augmented LRU page replacement algorithm shows better performance in terms of runtime.

VII. CONCLUSION

In our proposed system we used LRU page replacement algorithm with augmented doubly circular link list and hash table. We implemented the system and compared its performance with the classical LRU page replacement algorithm and found that our proposed system shows better performance than the classical LRU, i.e. the proposed LRU with augmented data structures will give a minimum of 50 percent improvement than the LRU using stack. The performance of LRU page replacement algorithm can be improved by using augmented data structures instead of developing new algorithms. The proposed system can be used for different applications such as operating systems, web cache, databases etc.

VIII. ACKNOWLEDGMENT

We are pleased to thank Mr. Senthil Kumar on his valuable comments, which improved the research clarity and conciseness. We extend our gratitude to our friends for their support.

REFERENCES

[1] Thomas H.Corman, Charles E. Leiserson et al., Introduction to Algorithms, 3rd ed. PHI publication, 2010

[2] Abraham Silberschatz, Peter B. Galvin and Greg Gagne,"Memory Management Strategies"in Operating System Concept,8th ed. Wiley Student Edition, pp. 315-417.

[3] Andrew S. Tanenbaum,"Memory Management,"in Modern Operating Systems,3rd ed. New Delhi, Pearson Prentice Hall, 2009, ch. 3, pp. 175-248.

[4] C.C.Kavar and S.S. Parmar, "Performance Analysis of LRU Page Replacement Algorithm with Reference to Different Data Structure," IJERA, 2013, Vol. 3, Issue 1, pp. 2070-2076.

[5] Lars Arge,"Efficient External-Memory Data Structures and Applications,"Ph.D. dissertation, faculty of science, university of Aarhus, Denmark, Feb 1996.

[6] Yannis Smaragdakis, Scott Kaplan and Paul Wilson,"The EELRU Adaptive replacement algorithm,"Elsevier,Performance Evaluation 53, 2003, pp. 93-123.

[7] S.M.Shamsheer Daula, Dr. K.E. Sreenivasa Murthy and G amjad Khan, "A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management," IJERA, vol. 2, pp. 126-130, March-April 2012.

[8] Norbert P. Kusters,Andrea D'Amato,Vinod R. Shankar,"Cache Employing Multiple Page Replacement Algorithms",United States Patent Application Publication, pub.no:US2013/0219125 A1, pub.date:Aug. 22, 2013

[9] Debabala Swain, Bijay Paikaray and Debabrata Swain, "AWRP: Adaptive Weight Ranking Policy for Improving Cache Performance," Journal of Computing, Volume 3, February 2011.

[10] Jaafar Alghazo, Adil Akaaboune and Nazeih Botros,"SF-LRU Cache Replacement Algorithm," MTDT IEEE Conference, 2004.

[11] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho and Chong Sang Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," IEEE Transactions on Computers, vol. 50, no. 12, Dec, 2001, pp. 1352-1360.