International Journal of Research In Science & Engineering e-ISSN: 2394-8299
Volume: 3 Issue: 4 July-August 2017 p-ISSN: 2394-8280

ENHANCED PAGE REPLACEMENT TECHNIQUES FOR DEMAND PAGING SYSTEM

Ayush Mittal[1], Shivaditya Jatar[2]

[1]*B.E. Scholar, Computer Engineering Department, SGSITS, Indore, ayushmi62@gmail.com*
[2] *B.E. Scholar, Computer Engineering Department, SGSITS, Indore, shivadityajatar@gmail.com*

ABSTRACT

The purpose of virtual memory is to enlarge the address space, the set of addresses a program can utilize. The operating system divides virtual memory into pages, each of which contains a fixed number of addresses. Each page is stored on a disk until it is needed & in an operating system that uses paging for virtual memory management, Demand Paging is used. Demand Paging's effectiveness depends on page replacement policy used as well as many other factors, but the project focuses on enhanced page replacement policies and their simulation. So, the first proposed technique for page replacement in demand paging is Advanced LRU. It uses some parameters like Spent-Time-Since-Last-Reference (T) and Total-Number-of-References (R), which provide better criteria to remove the page from TLB. Another technique has been proposed which makes the use of Augmented Data Structures. In this technique the data structure used in the TLB is augmented with others, so as to decrease the time for fetching the required page. This paper shows that the hit-rate and hit-ratio was significantly improved than the existing algorithms, thus proving the proposed techniques to be more efficient.

Keywords*: Augmentation, Demand Paging, Advanced LRU, Page Replacement.*

---

1. INTRODUCTION

As new pages need to be added to the temporary TLB, old ones need to be removed. As, this can affect the performance of the memory, the page that is removed due to demand paging should either increase the performance or have no effect at all. So, the page replacement policy that is designed here, aims to remove the page, that increases the hit-ratio [1].

For implementing new policies in the system, it is vital to measure its performance, comparing them with existing ones by measuring the efficiency of the researched page replacement policies, effects of thrashing, calculating cache hit or miss ratio, etc.

While the new Linux memory management is well-documented, it is typically difficult to fully understand the behavior of a Linux subsystem without direct experimentation, especially if planning to modify it in further development or research. This would typically involve observing the behavior of the memory management module in a variety of synthetic test cases, followed by an attempt to change the memory module to implement a new page replacement policy, where it is determined how those changes alter its behavior. The existing Linux kernel uses LRU as its page replacement policy, as it is the most suitable one in practical terms. Therefore Researchers are modifying and extending LRU policy with new ones. This Project aims to increase the performance of the replacement of page during demand paging by designing new algorithm that increases the hit-ratio and also to perform a comparative analysis with the existing algorithms, to see the performance of these new proposed algorithms in actual environment, and whether they will enhance the performance by decreasing page fault rate while acquiring minimum overheads [1], [2].

The predefined algorithms like FIFO, LRU, MRU, etc., have been tested in various conditions and also implemented in real life. And their behavior is known, but if some new algorithms need to be deployed, rather than deploying in an actual system, it could be first tested in a simulating environment.

There exist certain algorithms that have not yet been deployed but have been researched extensively. The implementation of these algorithms can be observed in the simulator to find loopholes in order to enhance the performance. The algorithms that have been researched and modified in this project are:

1. Advanced LRU Page Replacement Algorithm
2. Augmentation of Data Structures for improving LRU's performance.

The main problem is to design and implement page replacement techniques that are applied during demand paging, such that they increase the performance during this process. These techniques improve the hit-ratio and decrease the miss–rate incurred during the page replacement process. To also examine the effect of new policy with that of the existing ones, on the parameters of hit-ratio and hit-rate, by simulating the environment required for this.

2. BACKGROUND WORK

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. The existing Linux kernel uses LRU as its page replacement policy, as it is the most suitable one in practical terms [2]. Therefore Researchers are modifying and extending LRU policy with new ones [5]. The old page replacement mechanism in Linux (up to and including 2.6.27) has two major classes of problems:

- Sometimes the kernel evicts the wrong pages, resulting in bad performance
- The kernel scans over pages that should not be evicted. This results in increased CPU use on large memory systems, but results in catastrophic CPU use and lock contention on huge systems.

3. APPROACHES

1. Advanced LRU Page Replacement Algorithm
2. Augmentation of Data Structures for improving LRU's performance.

Advanced LRU Page Replacement Algorithm

The algorithmic steps to determine which page to evict after any page fault are as follows:
Parameters used :

R : Total number of references.
M : Modified bit (Set 1 if page is modified else 0, set 0 after eviction, each page table contains a Modified reference entry, initialized to zero at the start i.e. M =0, for all pages. When the contents of any page change, M will be set, M=1 for that page)
T : Spent time since last reference (for each page that holds the time when that page was referenced)

Step 1: When a page fault occurs, it will check (R) for all pages. If only one page found with minimum (R) value, that page will be evicted from memory.
Step 2: If there are more than 1 pages with minimum (R) value, M=0 for all of them, treat it same as LRU.
Step 3: If a page with minimum (R) value is found having, M=1 then select that page for eviction.
Step 4: If minimum (R) value is shared between no. of pages, and all of them have M=1, then evict the page which is recently modified considering the most (T) value among those pages.
Note: (R) and (T) values are checked every time for each page to select a page for eviction.

Augmentation of Data Structures (Doubly Linked Lists) in existing LRU policy

In doubly circular link list we apply the **move-to-front (MTF)** self heuristic method to design the LRU page replacement algorithm. Self arrangement improves the average access time in the link list. The goal of this augmentation in doubly circular link list is to improve efficiency of linear search [11].

MTF method moves the most recently accessed element to the head of the link list shown in figure 1. So, all the most recently accessed nodes are available at the head of the link list for future access.

When any node is accessed, it directly moves to head of the link list. Therefore most recently pages moves at head or near about it. Also we insert the element at the head position, so first element of self-adjustable doubly circular link list always most recently used. Therefore all the pages at the tail of the link list are least recently used. So we delete the element from the tail of the self adjustable doubly circular link list. Therefore this technique has no extra computational or memory overhead.

Algorithm :
Step 1: Search page table for page p.
Step 2: If page is available then move page p to the head of the self-adjustable doubly circular link list. Increase hit counter by one and exit; otherwise increase miss counter by 1 and go to step 3.
Step 3: If free space in page table then insert page p to the head of the link list otherwise go to step 4
Step 4: I) Delete the page from the tail of the link list then
II) Insert the page p at the head of the link list

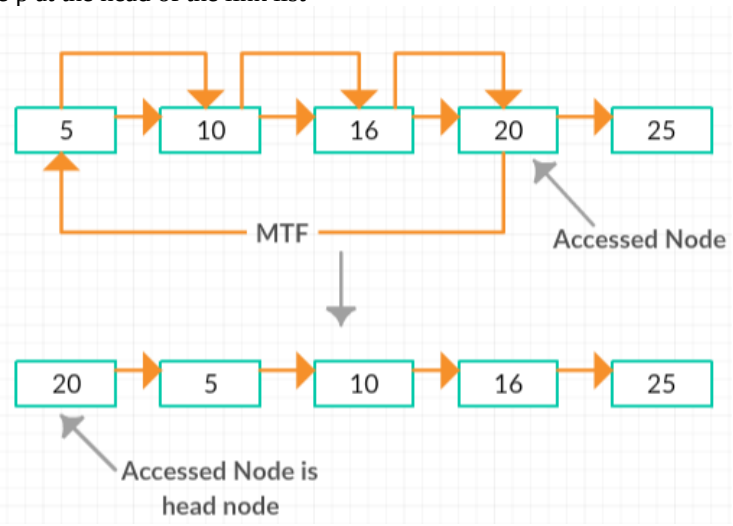

Fig-1: MTF method

4. SIMULATION AND RESULT ANALYSIS

We used the following two test cases to simulate our algorithms. Each test case contains the no. of processes and the no. of reference strings used.

Test Case 1: No. of Processes: 4 , No. of Reference Strings: 50
The time taken by LRU algorithm is 8.6 msec whereas that of Augmented LRU is 7.525 msec.

Algorithm	**No of page miss**
FIFO	35
LRU	33
Optimal	29
Advanced LRU	25
Augmented LRU	33

Table-1: Result of Test Case 1

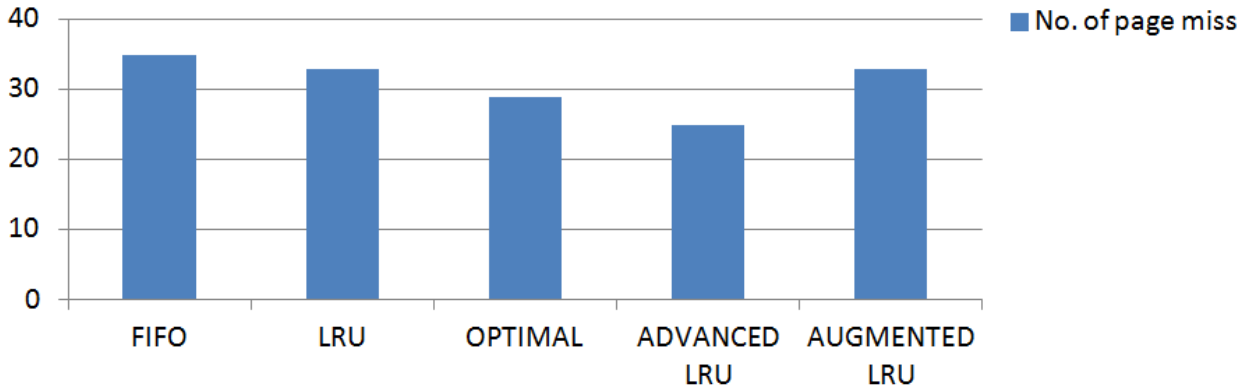

Fig-2:Graph of Test Case 1

Test Case 2: No. of Processes: 5, No. of Reference Strings: 100
The time taken by LRU algorithm is 9.0 msec whereas that of Augmented LRU is 7.765 msec.

Algorithm	No of page miss
FIFO	62
LRU	54
Optimal	49
Advanced LRU	42
Augmented LRU	54

Table-2: Result of Test Case 2

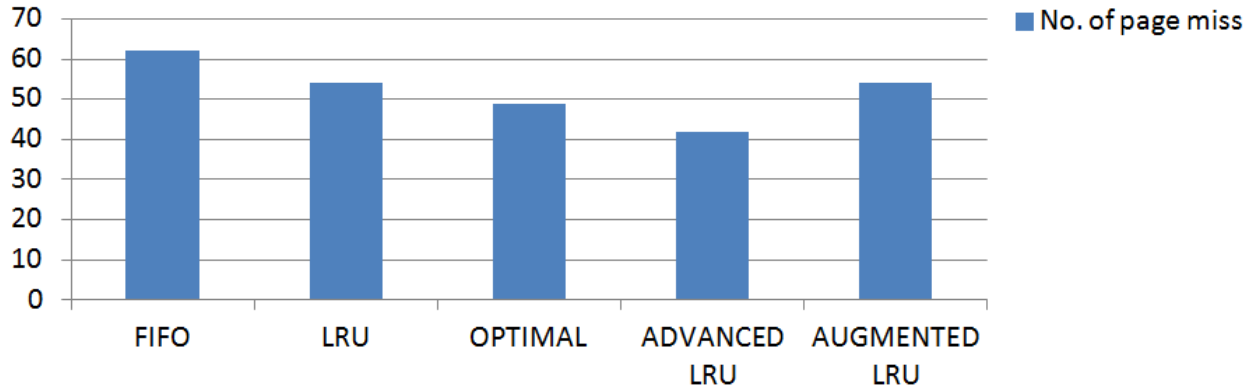

Fig-3:Graph of Test Case 2

5. CONCLUSION & FUTURE WORK

The new page replacement policies, i.e., Advanced LRU and Augmentation of data structures are implemented in a simulator on same data set, and the results are compared with the existing policies.

We can further incorporate Machine Learning and Data Analytics to understand patterns in a given program in order to automate generation of reference strings. Furthermore, augmentation of data structures like hash table, splay tree, etc can be done to improve the performance of page replacement algorithms.

REFERENCES

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

[2] Andrew S. Tanenbaum, “Memory Management,” in Modern Operating Systems,3rd ed. New Delhi, Pearson Prentice Hall, 2009, ch. 3, pp. 175248.

[3] Daniel Dominic Sleator and Robert Endre Tarjan, “Self-Adjusting Binary Search Trees,” Journal of the Association for Computing Machinery. Vol. 32, No. 3, July 1985, pp. 652-686.

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

[5] 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 Comput-ers, vol. 50, no. 12, Dec, 2001, pp. 1352-1360.

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

[7] Kaveh Samiee and GholamAli Rezai Rad, “WRP: Weighting Replacement Policy to Improve Cache Performance,” International Journal of Hybrid Information Technology, vol.2, 2009.

[8] 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.

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

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

[11] Wikipedia, (2017, April 20). Doubly linked list[Online]. Available: https://en.wikipedia.org/wiki/Doubly_linked_list

[12] Wikipedia, (2017, April 20). Demand paging [Online]. Avail-able: https://en.wikipedia.org/wiki/Demand_paging