American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-9, Issue-3, pp-193-197
www.ajer.org
Research Paper Open Access

Advancement and Analysis of Proficient Page Replacement Algorithm

[1]Enayet Kabir, [2]Md. Muniruzzaman, [3]Most.Masuma Akther,
Department of computer science and engineering, DUET, Bangladesh

[4] Md. Mahbub Alam
(National Merit Awarded by IDEB-2010) MSc in ECE, IU, Kushtia, Bangladesh
Corresponding Author: Md. Mahbub Alam

ABSTRACT : *In this paper we studied different page replacement algorithms and compared their performance. We proposed a new page replacement algorithm that uses the combination of sequential and LRU methods to obtain the better performance than that of various existing methods. We also give our attention to substitute some former method characteristic based on new idea. Key concept is that the replacement algorithm should reduce the page fault of system.*
KEYWORDS: *Main memory, cache memory, page replacement algorithms, reducing page fault.*

Date of Submission: 01-03-2020 Date of acceptance: 16-03-2020

I. INTRODUCTION

In computer operating systems, paging is one of the memory-management schemes by which a computer can store and retrieve data from secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. A page fault is a trap to the software raised by the hardware when a program accesses a page that is mapped in the virtual address space, but not loaded in physical memory.

Memory is an important resource that must be carefully managed. Since main memory is usually too small to accommodate all the data and programs permanently, the computer system must provide secondary storage to back up main memory. Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.

A **CPU cache** is a memory used by the central processing unit of a computer to reduce the average time to access memory. Cache memories are used in modern, medium and high-speed CPU to hold temporarily those pages of main memory .The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are cached in memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory. When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory. The data cache is usually organized as a hierarchy of more cache levels L1, L2, etc. In the memory hierarchy, position of cache is between processor and main memory (RAM). If the requested page is not present in cache, this event is called a cache miss or if present in cache, this event is called a cache hit . If the requested page is not present in main memory, this event is called a main memory miss or if present in main memory , this event is called a main memory hit .

We have studied different types of algorithms of page replacement such as FIFO, Optimal, LRU, NRU, Clock, LRU etc. In this paper we work to reduce the page fault as well as to reduce the overhead to system.

II. RELATED WORK

Cache memory is important for central processing unit, used to reduce the average access time of CPU. It is positioned in between the main memory and CPU. Each page replacement algorithm can reduce number of page faults and also may reduce overhead of the system. Another low-overhead page replacement algorithm is **FIFO** (**First-In First-Out**) algorithm. FIFO page replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced the oldest page is chosen. The most unexpected condition occurs in FIFO is Belady's anomaly[8].The politic considered in NRU, is that a page without any change, during its residence in primary memory, can be known as a desirable page to be expelled from the memory. This algorithm uses two status bits named Reference bit (R) and Modification bit (M). However it is unnecessarily inefficient, since it is constantly moving pages around on its list. But a better approach can be keeping all the page frames on a circular list, in a clock form. The old page will be pointed and when a page fault occurs then the pointed page will be investigated for page replacing operation. An optimal page replacement algorithm has the lowest page fault rate for any page reference stream, it simply replace the page that will not be used for the longest period of time. LRU replacement associates with each page the time of that page's last use.

III. LITERATURE SURVEY

A. The FIFO Page Replacement

According to this algorithm, when a page fault occurs, if there is one or more empty frame(s) in primary memory, the new invoked page will be loaded into it or one of them. But for cases in which no empty frames exist, a page with the most memory residence time will be selected for exit. In the other word, a page that has entered to the memory before all other pages, will exit the memory before them. The idea behind this politics is that, the first entered page has had enough chance to be used and this chance must be given to another page. This algorithm suffers from some disadvantages. As the first, if a page is used frequently in several time periods, it will be identified as the last or oldest

page, ultimately, and may be selected to be moved out from the memory, while there is a considerable probability for urgent need to it. In such these cases, the selection will be inefficient; since the removed page must be reloaded into memory almost immediately. Another disadvantage for this algorithm relates to this fact that increasing the memory frames designated for a process can yield to a lower page fault ratio; but Belady, Flone and Shelder found that, in using FIFO for special page invoking sequences, increasing the frames number will increase the page fault ratio. Such this event is referred to as FIFO Anomaly. Figure-1 and figure-2 depict the FIFO execution for different frames number.

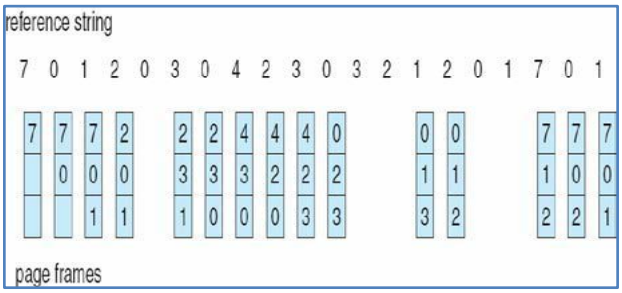

Fig-1: FIFO anomaly diagram

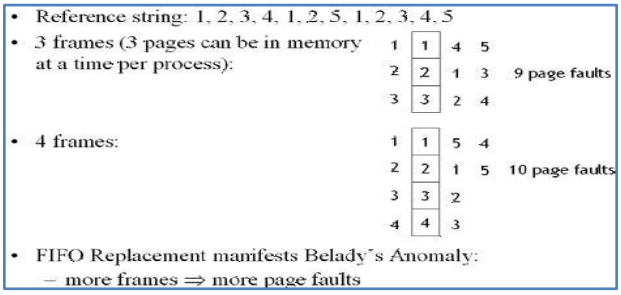

Fig-2: FIFO anomaly diagram

B. The NRU Page Replacement Algorithm

The politic considered in NRU, is that a page without any change, during its residence in primary memory, can be known as a desirable page to be expelled from the memory. This algorithm uses two status bits named Reference bit (R) and Modification bit (M). These bits are contained in each page table entry, and the algorithm initializes them with zero, for all pages. When a page is referred to or its contents change, R or M will be set, respectively. Since, these bits must be updated for each page referring, it is essential that they be set by the hardware. When there is a need to replace a page with a new one, first, it is attempted to find a page without any reference (R=0) . If no such this page was found, a page with R=1, preferably with M=0 (without change) will be selected. The reason for such this selection is that removing pages with change (M=1), impose a secondary memory rewriting overhead. Since the reference bit for most pages is set to one, gradually, the ability to detect the most appropriate page, for exiting the primary memory, will be reduced. To preventing such this challenge, all pages reference bits are reset periodically. Regarding different states for R and M, four major groups of pages can be imagined:

Class 0: Not Referenced, Not Modified (R=0, M=0).

Class 1: Not Referenced, Modified (R=0, M=1).

Class 2: Referenced, Not Modified(R=1, M=0).

Class 3: Referenced, Modified(R=1, M=1).

The groups with lower numbers include the pages with more priority to exit the memory, and the later group's members have minimum priority. The contradictory state for class 1 is via the

C. The Clock Algorithm

The second chance algorithm could be known as a reasonable algorithm. However it is unnecessarily inefficient, since it is constantly moving pages around on its list. But a better approach can be keeping all the page frames on a circular list, in a clock form. As can be seen in figure the oldest page is pointed by a hand. When a page fault occurs, the page being pointed to by the hand is investigated. If its reference bit (R) is 0, the page is moved out of memory, and is replaced with the new page. Then the hand is advanced one position. Otherwise, if R=1, it is cleared and the hand is advanced to the next page. This process is repeated until a page with R =0 is found. This algorithm, so called *clock*, differs from second chance algorithm, only in the implementation.

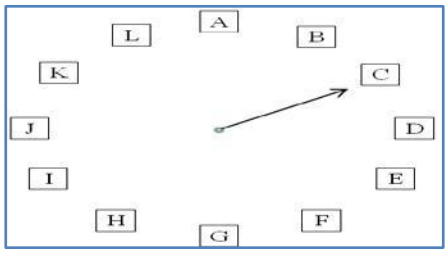

Fig: Clock Algorithm

D. The Optimal Page Replacement Algorithm

An optimal page replacement algorithm has the lowest page fault rate for any page reference stream, it simply replace the page that will not be used for the longest period of time. The problem here is to know the future perfectly. Figure- 4 shows an example of this algorithm.

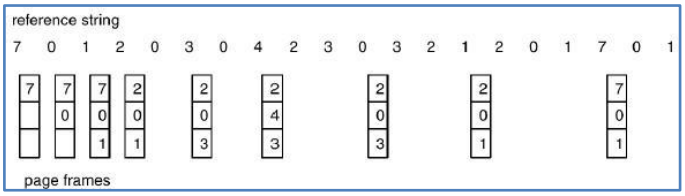

Fig: The Optimal Page Replacement Algorithm

E. The LRU Page Replacement Algorithm

LRU replacement associates with each page the time of that page's last use. A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions

will probably be heavily used again in the next few. Conversely, pages that have not been used for pages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called Least Recently Used (LRU) paging page replacement algorithm. Figure.5 depict the LRU execution for different frames number.

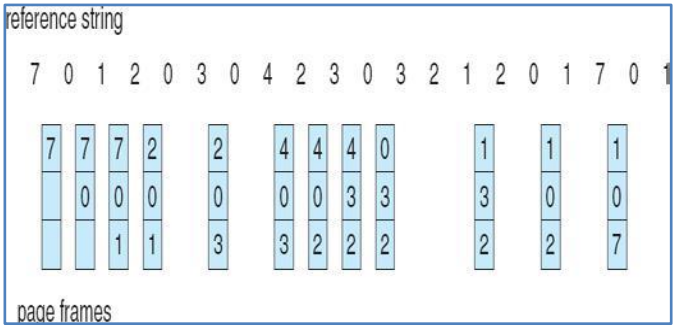

Fig: The LRU Page Replacement Algorithm

IV. PROPOSED APPROACH

We propose a new algorithm for page replacement. The technique is as follows:

- If the requested page is not available in main memory and also not available in cache memory then page fault will occur .In this situation the page will be loaded from secondary storage to main memory and also keeps a copy in cache memory. In our methodology, loading of page follows the sequential manner for both of main memory and cache memory, until they become full. Only when the cache memory is full and page fault occurs the page replacement in cache memory uses LRU algorithm.

- If the page is available in main memory (no page fault occur) the page is stored in a temporary variable first and then the page of cache memory using the LRU algorithm will send to the last location of buffer in main memory if it is not available in main memory (when the frame array is full). At the last step, the content of temporary variable will be stored in cache memory in the position from where the page number has been brought for buffering in main memory. If continuously it occur then brought page also stored in continuously at the last position until the last occur and also pointer of main memory buffer will not be set to initial in that cases. If continuously not occur after execution of above works the pointer of buffer of main memory will be reset to the initial position of the buffer. Thus the process will be continued.

A. Proposed Algorithm

1. Input reference string size , main memory and cache memory size ; // “n,f,c are reference string length , //main memory frame size and cache memory frame // respectively.”
2. Input reference string in ref[] ; // “ ref[] , mm[],cm[] are //arrays for reference string , main memory ,cache //memory respectively.”
3. //u,v,g,s,ml,cl,pf are temporary variable
4. //ml=main memory location, cl= cl=cache memory location
5. For i=0 to n-1 do
6. Set s:= required page.
7. Find page fault for required page
8. Search page in main memory
9. Search page in cache memory

B. Description of Proposed Approach with example

Let, size of cache memory is =3
Size of main memory =6
Reference string: 1 2 3 4 5 6 7 8 1 2

From step 1 to 3 the page fault occurs and the pages stored sequentially both in main memory and cache memory.

In step 4 to 6 when the page fault occurs the page has been replaced in cache memory following LRU and stored in main memory sequentially.

In step 7 and 8 when the frame numbers are full the requested page will be loaded in last position and this process will continue till page fault occur.

In step 9 and 10 the requested page is available in main memory so there is no page fault and the page will be stored in a temporary variable first and then the page of cache memory following the LRU algorithm will be stored in the last location of frame array in main memory (when the frame array is full). At the end, content of temporary variable will be stored in the position from where the page number has been transferred from the cache memory to main memory. After execution of all steps the pointer of frame array will reset to the initial position of the memory buffer. Thus the process will be continued

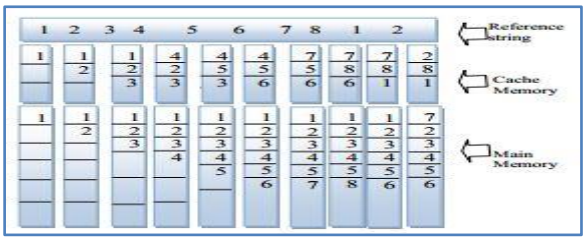

Fig: Proposed Algorithm

C. Comparison between LRU and Newly Approached Algorithm

Reference String :1 2 3 4 5 6 7 8 1 2		
Frame number (Main memory)=6 Cache memory frame number=3		
LRU		**Our Approach**
Total number of page		Total number of page
fault=10		fault=8

Fig: LRU and Newly Approached Algorithm

V. CONCLUSION

In this paper, we have proposed a new page replacement algorithm, which is essentially a modified version of the celebrated LRU algorithm. The basic distinction of our approach with the existing page replacement algorithm is that in the new one, the pages in cache and RAM always remain synchronized, whereas, in the former, they remain a synchronized We have shown that the page fault rate in our developed algorithm is much lower than that of LRU and thus it performs better.

REFERENCES

[1]. John S. Liptay. Structural aspects of the system/360 model 85 ii: The cache. IBM Systems Journal, 7(1):15-21, 1968.
[2]. Gururaj S. Rao. Performance analysis of cache memories. J. ACM, 25(3):378-395, 1978
[3]. William D. Strecker. Cache memories for pdp-11 family computers. In ISCA, pages 155-158, 1976
[4]. Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall, 1992.
[5]. Hongliang, G., Chris, W., "A Dueling Segmented LRU Replacement Algorithm with Adaptive Bypassing", JWAC 2010 - 1st JILP Worshop on Computer Architecture Competitions: Cache Replacement Championship (2010), pp. 213 – 216, 2010.
[6]. Ali Khosrozadeh, Sanaz Pashmforoush, Abolfazl Akbari, Maryam Bagheri, Neda Beikmahdavi "A Novel Page Replacement Algorithm Based on LRU " *J. Basic. Appl. Sci. Res.*, 2(10)10377-10383, 2012
[7]. Operating System Principles(6th edition) Abraham Silberschatz, Peter Bare Galvin, Greg Gagne
[8]. Modern operating system (2nd edition) Andrew S.Tanenbaum
[9]. Operating Systems: A Concept-based Approach (2E) D.M. Dhamdhere

Md. Mahbub Alam, etal " Advancement and Analysis of Proficient Page Replacement Algorithm ". *American Journal of Engineering Research (AJER)*, vol. 9(03), 2020, pp. 193-197.