Volume 6, Issue 1, January 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Enhanced Hybrid LRU Page Replacement Algorithm

K. Ravindra Babu*, G. Pavan Kumar
Department of Computer Science and Engineering,
Kamala Institute of Technology and Science, Telangana, India

Abstract—whenever a page fault occurs in main memory, the operating system has to choose a page to remove from main memory in order to make a space for the new page to be brought in to the main memory in order to continue the execution of a large program. If the page that is to be removed from memory has already been modified while residing in the memory, must be written to the disk in order to bring the disk back up copy up to the date. However, if the page has not been modified, then the disk copy is already up to the date, so there is no need to write the page in to the disk. The page that is to be read in just overwrites the page being removed from the main memory. Many algorithms have been proposed for replacing a page from the main memory to the disk. A page replacement technique is called efficient if it produces minimum number of page faults. The main goal of these techniques is to minimizing the page faults occurrences as much as possible and reducing the cost of removing a page.

Keywords—Page Replacement, Page Fault, FIFO, Optimal, LRU, CLOCK, Counting Based Page Replacement, ARC, LRU-K, LIRS

I. INTRODUCTION

The most important part of the operating system is memory management. Main memory is divided into fixed size units called page frames. Each victimized page can be either in secondary memory or in main memory as page frames. The virtual address space is divided into fix size blocks called pages and this division is done by operating system.A CPU generated address is called logical address or virtual address, whereas memory management unit generated address is known as physical address. Before using this logical address, it must be translated to its corresponding physical address. This address translation has been done corresponding to every memory reference, so it is important that it must be fast. A special hardware unit referred to as Memory Management Unit (MMU)is used for such translation. MMU uses address mapping information which is usually located in page tables, to make the translation. If the given virtual address is not mapped to main memory, operating system is trapped by the MMU. This trap is called as page fault which gives an opportunity to the operating system to bring the desired page from secondary memory to main memory, and then update the page table correspondingly. In simple words we can say that-When the processor need to execute a particular page and main memory does not contain that page, this situation is known as PAGE FAULT. As each and every process has its own virtual address space, the operating system must keep track of all pages and the location of each page used by each process. Whenever any page in main memory is referenced or written to, it is marked accordingly. When a page fault occurs (known as cache miss), the operating system eliminates some page to secondary memory to make space for the incoming page. Whenever a cache miss occurs, the operating system applies page replacement algorithm to choose a page from cache for replacement or eviction to make place for the referenced page. Virtual memory system requires effective page replacement algorithms in case of a page fault, to decide which pages should be expelled from memory. Since long many algorithms have been proposed for page replacement. Each algorithm tries to reduce the page fault rate while acquiring minimum overhead.

II. PREVIOUS WORK

A. page replacement algorithms

Page replacement takes the following approach. If no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory [1]. We can now use the freed frame to hold the page for which the process faulted.

Basic Page replacement goes as follows:

1. Find the location of the desired page on the disk.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

Various page replacement algorithms explained through this paper are as follows:

1) First-In-First-Out (FIFO)

The simplest page-replacement algorithm is a FIFO algorithm (first in first out). The first-in, first-out (FIFO) [8] page replacement algorithm is a less-overhead algorithm that entails little book-keeping on the part of the operating system. The idea is obvious from the name - the operating system keeps track of each page in memory in a queue, with the latest arrival at the back, and the earliest arrival in front. The operating system sustains a list of all pages presently in memory, with that page which is at the head of the list the oldest one and the page at the tail the most topical arrival. When a page needs to be swapped, the page at the front of the queue (the oldest page) is considered. While FIFO is cheap and instinctive, it results poorly in practical application.

2) Optimal Page Replacement

One result of the discovery of Belady's anomaly was the search for an optimal page-replacement algorithm which has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly [1]. Such an algorithm does exist and has been called OPT or MIN. It is simply: Replace the page that will not be used for the longest period of time. Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

3) Least-Recently-used (LRU)

This algorithmreplaces the page that has not been used for the longest period of time. We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward [1]. FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used. If we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time. The LRU policy is often used as a page-replacement algorithm and is considered to be good. The major problem is how to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance. The problem is to determine an order for the frames defined by the time of last use.

4) Clock Algorithm

In CLOCK, all page frames are visualized to be arranged in form of a circular list that resembles a clock. A pointer (that is, a hand on the clock) indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a 0 reference bit. As it advances, it clears the reference bits. Once a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position. Notice that, in the worst case, when all bits are set, the pointer cycles through the whole queue, giving each page a second chance. It clears all the reference bits before selecting the next page for replacement. Second-chance replacement degenerates to FIFO replacement if all bits are set [1].

5) Second-Chance Algorithm

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, we inspect its reference bit. If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page [1]. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced or given second chances. In addition, if a page is used often enough to keep its reference bit set, it will never be replaced.

6) Enhanced Second-Chance Algorithm

We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair [1]. With these two bits, we have the following four possible classes:

(0, 0) neither recently used nor modified -best page to replace

(0, 1) not recently used but modified-not quite as good, because the page will need to be written out before replacement

(1, 0) recently used but clean-probably will be used again soon

(1, 1) recently used and modified -probably will be used again soon, and the page will be need to be written out to disk before it can be replaced

Each page is /in one of these four classes. When page replacement is called for, we use the same scheme as in the clock algorithm; but instead of examining whether the page to which we are pointing has the reference bit set to 1, we examine the class to which that page belongs. We replace the first page encountered in the lowest nonempty class. Notice that we may have to scan the circular queue several times before we find a page to be replaced.

7) Adaptive Replacement Cache (ARC)

The Adaptive Replacement Cache (ARC) algorithm [2] basically provides an improvement over LRU based algorithms by taking two factors: the recency of pages and frequency of occurrences of pages into account. These two factors are accomplished by maintaining two lists, L1 and L2 and by remembering the history of the pages. These two lists together make a directory called as cache directory. The list L1 is used to capture the recency of pages and the list L2 to capture the frequency of occurrences of pages. The sizes of both the lists (L1 and L2) are kept at roughly as that of the size of the number of page frames in the main memory, say 'm'. Therefore, the history of at most m pages, not in the main memory, has to be remembered. The lists L1 and L2 are partitioned into two lists, T1, B1, and T2, B2.

Where: T1 contains in-cache pages that have been accessed only once, and T2 pages that have been accessed more than once, while on lists. Likewise, list B1 stores the history of pages evicted from the list T1, and B2 stores the history of pages evicted from the list T2.

8) Low Inter-reference Recency Set (LIRS)

The Low Inter-reference Recency Set algorithm takes into consideration the Inter-Reference Recency of pages as the dominant factor for eviction [3]. The Inter-Reference Recency (IRR) of a page refers to the number of other pages accessed between two consecutive references to that page. It is assumed that if current IRR of a page is large, then the

next IRR of the block is likely to be large again and hence the page is suitable for eviction as per Belady's MIN. It needs to be noted that the page with high IRR selected for eviction may also have been recently used. The algorithm distinguishes between pages with high IRR (HIR) and those with low IRR (LIR). The number of LIR andHIR pages is chosen such that all LIR pages and only a small percentage of HIR pages are kept in cache [4]. Now, in case of a cache miss, the resident HIR page with highest recency is removed from the cache and the requested page is brought in.

9) Counting-Based Page Replacement

There are many other algorithms that can be used for page replacement. For example, we can keep a counter of the number of references that have been made to each page and develop the following two schemes.

The **least frequently used (LFU) page-replacement algorithm** requires that the page with the smallest count be replaced [1]. The reason for this is that an actively used page should have a large reference count. A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

The **most frequently used (MFU) page-replacement algorithm** is based on the argument that the page with the smallest count was probably just brought in and has yet to be used [1].

10) LRU-K method

The LRU-K [5] policy takes into account the recency information while evicting pages from memory. It does not consider the frequency factor. Because when taking the frequency factor in to consideration then this algorithm will evict the pages with the largest backward K- distance. The Backward K-distance of a page p is referred to as the distance measured from backward, means from the current time to the Kth most recent reference to the page p. Since this algorithm considers Kth most recent reference to a page, it will favor the pages, which are accessed frequently within a short time. As the value of K becomes higher, then it will not result in an appreciable increase in the performance, but in turn has high implementation overhead.

11) Page-Buffering Algorithms

Other procedures are often used in addition to a specific page-replacement algorithm. For example, systems commonly keep a pool of free frames. When a page fault occurs, a victim frame is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out [1]. This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written out, its frame is added to the free-frame pool.

An expansion of this idea is to maintain a list of modified pages. Whenever the paging device is idle, a modified page is selected and is written to the disk. Its modify bit is then reset. This scheme increases the probability that a page will be clean when it is selected for replacement and will not need to be written out.

Another modification is to keep a pool of free frames but to remember which page was in each frame. Since the frame contents are not modified when a frame is written to the disk, the old page can be reused directly from the free-frame pool if it is needed before that frame is reused [1]. No I/O is needed in this case. When a page fault occurs, we first check whether the desired page is in the free-frame pool. If it is not, we must select a free frame and read into it.

12) Pro- LRU

In this section, a LRU based algorithm is introduced, and is referred to as PRO_LRU [6]. The first and most important note about this algorithm is using an extra feature TNR (Total no. of references), which is to count total number of references and for selecting the outgoing pages, together with this feature, an extra feature is also used by LRU which is referred as STR (spent time since last reference). Whenever any page fault takes place, the first parameter, TNR, will be investigated for all pages. If there is only a single page with minimum TNR value, this page will be evicted from memory. Or else, if the minimum TNR value is divided between no. of pages, the second parameter comes on existence. In the other word, a page with most STR values with minimum TNR value will be selected for eviction.

13) Hybrid LRU

Hybrid LRU [7] also referred as HLRU is also an extension of LRU. It also uses the extra feature TNR (total number of references) for each encountered page and as a modification it uses the concept of modified reference, i.e. when a page is modified, a modified reference, M=1 will always be set for that page. When a page fault occurs, it examines the first parameter, TNR for each page. If only one page is found, it immediately replaces it. Else if there is more than one page is available with minimum TNR, it checks modified reference for those pages and replaces the page which is recently modified i.e. M=1. Each time when a modified page is re-modified all modified references for each page will be set to 0 by default.

III. PROPOSED ALGORITHM

A. Enhanced Hybrid LRU

The proposed algorithm is based on LRU and will be referred to as Enhanced Hybrid LRU, is thecombination of Hybrid LRU and Enhanced second chance algorithm. This algorithm uses an extra feature STR (spent time since last reference) for each page that holds the time when that page was referenced [6].

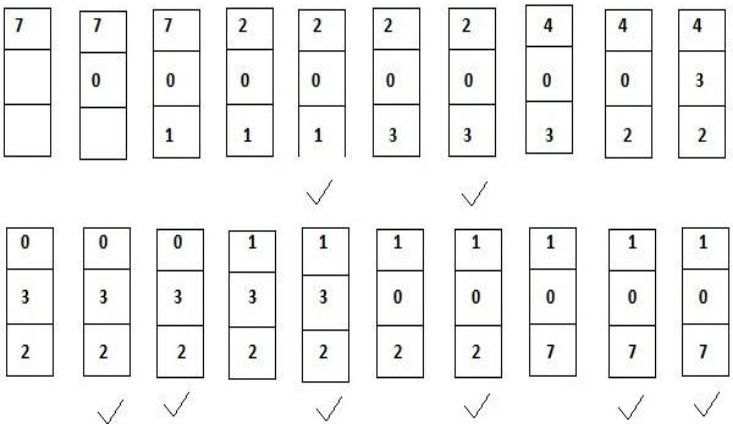
The algorithm goes as follows:

Step1: When a page fault occurs, it will check the first parameter, TNR for all pages. If only one page found with minimum TNR value, that page will be evicted from memory.

Step2: Modified reference is contained in each page table entry, and the algorithm initializes it with zero i.e. M =0, for all pages. When the contents of any page change, M will be set, M=1 for that page.

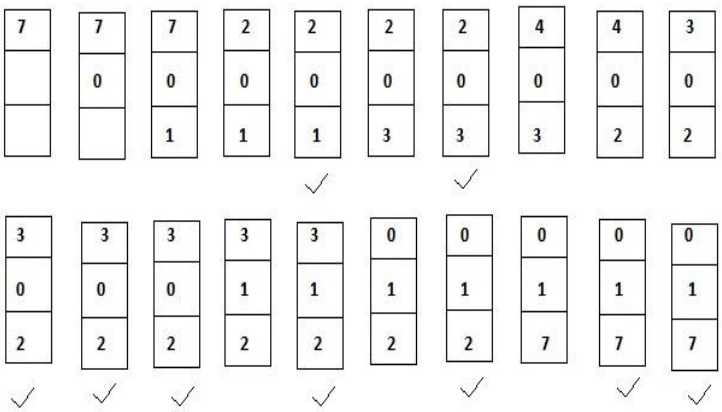
Step3: If the minimum TNR value is shared between no. of pages, R=0 and M=0, treat it same as LRU.
Step4: If a page with minimum TNR value is found having R=0, M=1 then select that page for eviction.
Step 5: If minimum TNR value is shared between no. of pages with R=0, M=1 then evict the page which is recently modified considering the most STR value among those pages.
Note: Whenever a modified page will be replaced again, modified bits for each table entry will be set to 0 i.e., M=0.
Step6: If a page having minimum TNR is found with R=1, M=0 and most STR value, that page is evicted.
Step7: If minimum TNR is shared between no. of pages with R=1, M=0, evict the page with most STR value.
Step8: If a page having minimum TNR value with R=1, M=1 is shared between no. of pages then repeat step 3 to 7until any page from any of above classes is found and until all pages complete their traversing.
Note: TNR and STR values are checked every time for each page to select a page for eviction.

IV. RESULTS



Page faults=12

Fig. 1 LRU representation of 3 frames



Page faults=11

Fig. 2 Enhanced Hybrid LRU representation of 3 frames

V. CONCLUSION

Previously page replacement algorithms like FIFO, OPT, ARC, Clock, LRU, and LRU-K, have implemented and compared them to evaluate their efficiency. Studying the page replacement algorithms has suggested that the LRU has had the best results among all practical algorithms, and the upcoming researches have focused on this algorithm, to increase the performance of system and to decrease the page fault rate. In this paper a page replacement algorithm based on LRU is represented. This algorithm uses a complementary parameter substitutingthe LRU parameter to determine the page i.e. TNR (total no. of references) that must be replaced with a new one with the concept of modified reference and STR (spent time since last reference).

REFERENCES

[1] Abraham, Silberschatz, Peter, Baer Galvin, & Greg, Gagne,Operating Systems Concepts 7th ed.,Wiley 2005.
[2] Gideon, Glass, & Pei, Cao. Adaptive-Page-replacement-based-on-Memory-Reference- behavior. Retrieved in extended form as Technical Report 1338 from www.cs.wisc.edu
[3] Song, Jiang, &Xiaodong, Zhan. LIRS: a Low Inter Reference Recency Set Replacement, SIGMETRICS2002.
[4] Song, Jiang, &Xiaodong, Zhang.. In Proceeding of 2002 ACM SIGMETRICS, LIRS: An Efficient Low Inter reference Recency Set Replacement Policy to Improve Buffer Cache Performance, 2002, June.

[5] Elizabeth, J. O'Neil, Patrick, E. O'Neil, & Gerhard, Weikum. In Proceedings of the ACM SIGMOD Conference on the The LRU-K Page Replacement Algorithm For Database Disk Buffering 1993.

[6] Ali Khosrozadeh, SanazPashmforoush, AbolfazlAkbari,MaryamBagheri, Neda Beikmahdavi., "Presenting a Novel Page Replacement Algorithm Based on LRU" , Journal of Basic and Applied Scientific Research , 2(10)10377-10383, 2012.

[7] Pooja khulbe, Shruti pant, "HYBRID LRU Page Replacement Algorithm" , International Journal of Computer Applications (0975 – 8887) Volume 91 – No.16, April 2014

[8] Amit S. Chavan, Kartik R. Nayak, Keval D. Vora, Manish D. Purohit and Pramila M. Chawan, "A Comparison of Page Replacement Algorithms" , IACSIT International Journal of Engineering and Technology, Vol.3, No.2, April 2011

[9] Operating system tutorial Available on:http://www.tutorialspoint.com/operating_system/os_virtual_memory.html

[10] Glass, G., & Cao, P. (1997, May). In Proceedings of 1997 ACM SIG- METRICS Conference on Adaptive Page Replacement Based on Memory Reference Behavior, (pp. 115-126)

[11] William Stallings. Operating Systems Internal and Design Principles 5th ed., Pearson 2006.

