COUNT BASED PAGE REPLACEMENT TECHNIQUE

SHREYANK N GOWDA

Department of Computer Science And Engineering, R.V.College Of Engineering, Bangalore, Karnataka, India
E-mail: kini5gowda@gmail.com

Abstract- It is important for a virtual memory system to be able to efficiently replace a page when a page fault occurs. Over time many algorithms have been developed to decrease time needed to make a decision whilst also minimizing overhead required to perform this. Here a theoretical approach is developed which like the optimum replacement technique can be used as a standard to compare the different existing algorithms. 'n' pages are calculated and 'n' variables are created to keep a count on the number of times each page is called. Each time a fault occurs we replace the page having minimum value of count among all the existing variables.

Index Terms- Page Replacement, LRU, Count, Page Fault

I. INTRODUCTION

In order to maximize the throughput of multiprogramming systems it is necessary accommodate more programs in the memory to perform interleaved execution. This will enhance the resource usage whilst also giving a better throughput as output. There are 2 types of memory- primary which is costlier but performs better and secondary which is cheaper but takes more time to perform it's operations. Virtual memory systems use this hierarchy to bring parts of a program into main memory from the secondary memory in terms of units called as pages [3]. Pages are brought into memory only as and when required, this is called demand paging. A page fault occurs when the requested page is not in secondary memory and hence needs to be brought out of the primary memory. To perform this one of the existing pages has to be removed. To perform this page replacement algorithms are implemented, which ensure minimum page fault rate. This paper gives an algorithm that is purely theoretical in nature and can be used as a standard henceforth to compare the efficiency of all other page replacement algorithms. It is similar to the optimal page replacement technique in the sense that both are purely theoretical in nature but act as standards.

II. BACKGROUND

Before demonstrating the algorithm and it's effectiveness it is important to recollect the major page replacement algorithms already in existence[7],[5]. Before we begin this discussion let us recollect some important terms, The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things. If we have a reference to a page p, then any immediately following references to page p will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.

1. Belady's Optimal Page Replacement Algorithm: This algorithm guarantees best efficiency. It works by removing the page from memory which will not be used for a long time. This optimal result is referred to as Belady's MIN algorithm or the clairvoyant algorithm. This algorithm needs to know the future in the sense that it should know beforehand which page will be least used in the future. Hence this algorithm is deemed inefficient or impractical for real life systems. This algorithm could be used to compare the effectiveness of other replacement algorithms by providing a lower bound on page fault rate. An example can be seen in Fig. 1.

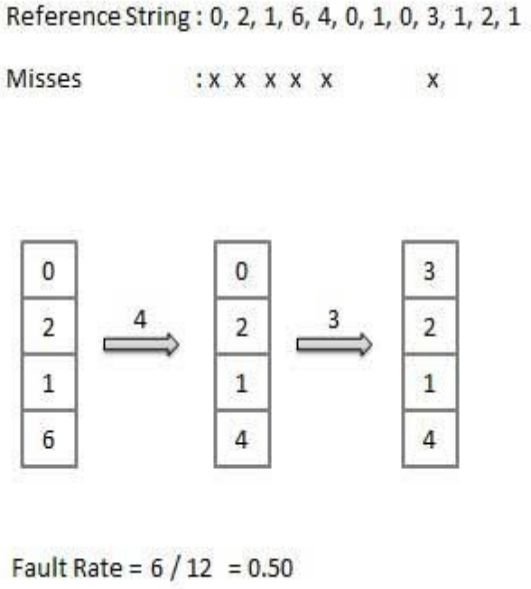


Fig. 1. Example for optimal replacement

2. **Least Recently Used (LRU) Page Replacement Algorithm:** Page which has not been used for the longest time in main memory is the one which will be selected for replacement. This is based on the principal of locality that states that program and data references within a process tend to cluster[8] . Easy to implement, keep a list, replace pages by looking back into time.

This uses the fact we already have the history of the page calls and assumes that since a page has been recently used would mean that we would not need to use it again any time soon. This is a wrong assumption however and could increase the page fault rate if there are repeated alternate calls to
a page that has just been replaced, then the number

of faults will increase massively. An example can be seen in Fig. 2.

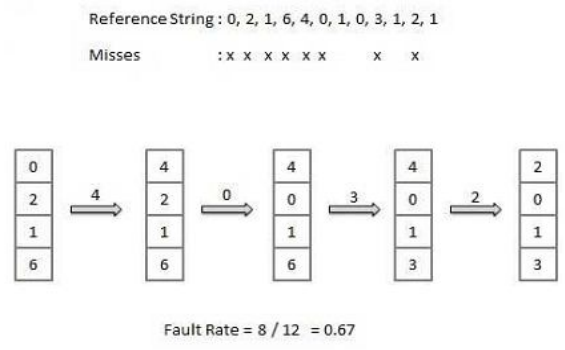


Fig. 2. Example for LRU replacement

3. **First In First Out Algorithm (FIFO):** This is the most simplest algorithm. Oldest page in main memory is the one which will be selected for replacement. Easy to implement, keep a list, replace pages from the tail and add new pages at the head. An example can be seen in Fig. 3.
4.

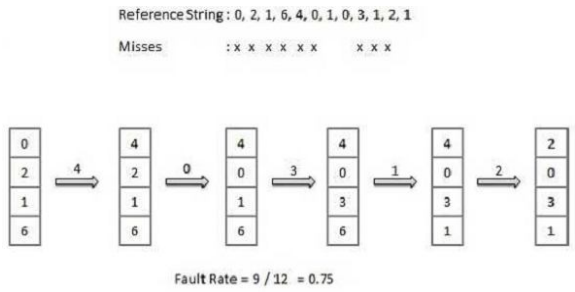


Fig. 3. Example for FIFO replacement

5. **Most Frequently Used Algorithm:** This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used. Count variables are maintained for each page and the page with the maximum count is replaced.
6. **Least Frequently Used Algorithm:** Page with the smallest count is the one which will be selected for replacement. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.
7. **LRU-K:** The LRU takes into picture how recently a page was used but does not consider freuqency. To consider the frequency details, LRU-K was proposed which removes pages with the largest backward K-distance. Backward K-distance of a page p is the distance backward from the current time to the Kth most recent reference to the page p. Since this policy considers Kth most recent reference to a page, it favors pages which are accessed frequently within a short time. Experimental results indicate that LRU-2 performs better than LRU [4]; higher K does not mean better performance, but does mean higher overhead.
Other complex algorithms include CAR attempts to join the adaptive policy of ARC with the execution efficiency of CLOCK [1]. The algorithm maintains four doubly linked lists T1, T2, B1, and B2. T1 and T2 are CLOCKs while B1 and B2 are simple LRU lists. Dueling CLOCK is an adaptive replacement policy that is based on the CLOCK algorithm. A major disadvantage of the regular CLOCK algorithm is that it is not scan resistant (The page replacement algorithm that does not allow scanning to push frequently used pages out of main memory is said to be scan resistant [2]). To combat this drawback, a scan resistant CLOCK algorithm has also been developed.

III. PROPOSED ALGORITHM

Proposed algorithm is as follows, first we determine number of pages. Let's say this is denoted by the value 'n'. Now we make 'n' count variables, say c1,c2, c3 upto cn. Now we take the reference string and count each value and add it to the count of that corresponding value. For example if our reference string value is 1,1,3,2,0,5,6,2,4 then we have 6 count variables and their values are, c1=2,c2=2.c3=1,c4=1,c5=1,c6=1 and c0=1.
Now say we have 4frames, so first 1 is entered to the frame and c1 is now equal to 1. Next 1 is already there, hence only c1 value changes and is equal to 0. Next c3 is made 0 and 3 is added. Similarly 2 and 0 are added. Now when 5 is to enter, one page fault occurs. To remove the value with minimum count is removed. If an ambiguous case occurs then the LRU algorithm or FIFO can be followed to remove a page. An example can be seen in Fig. 4.

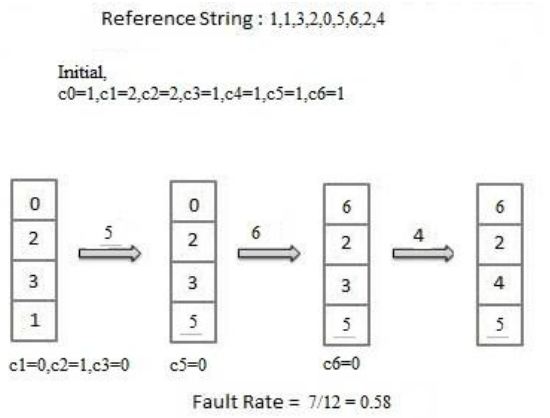


Fig. 4. Example for executed algorithm

The workflow of the algorithm is shown in Fig. 5,

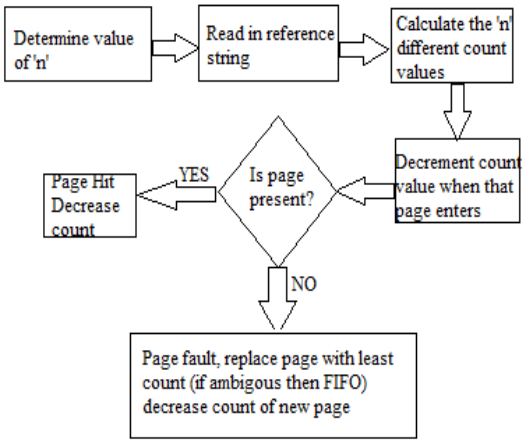


Fig. 5. Workflow for proposed algorithm

First read in the value of 'n', then read in the reference string. Count all the different number of occurrences of the page and make a count for each page. When a page is read decrease count of that page. Next we have to check number of frames. Initially till all the frames are full, page faults occur. Now when a new page enters we have to decide if that page is present or not. If it is, then that is a page hit and we should decrease the count of the page. If it is missed then it is a page fault. We have to replace one page from the frame. To do this we should decide based on the count of all pages in the frames. Repeat this process till we reach the end of the reference string. Then calculate the total fault rate as fault rate = misses/number of pages in reference.

IV. EXPERIMENTAL ANALYSIS

3 reference strings are taken and for each all the algorithms are executed and page fault is calculated. This can be seen as comparison of all the algorithms.

Test Case 1: Reference string: 1,2,3,4,7,6,5,1,3,1,2
Total number of frames: 4

Algorithm Name	Fault Rate
FIFO	0.83
LRU	0.83
Optimal	0.58
MFU	1.00
LFU	0.91
Proposed algorithm	0.58

Table 1. Test Case 1 comparison

Test Case 2: Reference string: 1,4,6,5,3,1,2,1,4,4
Total number of frames: 4

Algorithm Name	Fault Rate
FIFO	0.80
LRU	0.80
Optimal	0.60
MFU	0.80
LFU	0.80
Proposed algorithm	0.60

Table 2. Test Case 2 comparison

Test Case 3: Reference string: 1,3,2,4,5,2,5,1
Total number of frames: 4

Algorithm Name	Fault Rate
FIFO	0.75
LRU	0.75
Optimal	0.62
MFU	0.75
LFU	0.75
Proposed algorithm	0.62

Table 3. Test Case 3 comparison

CONCLUSION

As can be seen in the analysis the page fault rate comes exactly the same as Optimal even though they are both different algorithms. The only similar aspect of the algorithms is that they both need to know the future aspects of the reference string and hence cannot be practically implemented.

Hence it cannot be used in real life systems but can only be used as a standard to compare other algorithms.

The efficiency of the algorithm is excellent however and hence should be used as a standard and in particular it can be used as a standard for other counting based page replacement algorithms.

ACKNOWLEDGEMENT

I would like to thank the Head Of Department of the Computer Science Department Dr. Shoba G for her invaluable support and guidance. I would also like to thank my friends for their valuable inputs towards this idea. Last but not the least I would like to thank my parents and brother who have stood as an emotional boulder and supported me through everything.

REFERENCES

[1] S. Bansal, and D. Modha, “CAR: Clock with Adaptive Replacement”, FAST-’04 Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 187-200, 2004.

[2] A. Janapsatya, A. Ignjatovic, J. Peddersen and S. Parameswaran, “Dueling CLOCK: Adaptive cache replacement policy based on the CLOCK algorithm”, Design, Automation and Test in Europe Conference and Exhibition, pp. 920-925, 2010

[3] A. S. Sumant, and P. M. Chawan, “Virtual Memory Management Techniques in 2.6 Linux kernel and challenges”, IASCIT International Journal of Engineering and Technology, pp. 157-160, 2010.

[4] J. E. O’neil, P. E. O’neil and G. Weikum, “An optimality Proof of the LRU-K Page Replacement Algorithm”, Journal of the ACM, pp. 92-112, 1999.

[5] 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 pp. 171-174

[6] Jisha.P. Abraham, Sheena Mathew " A novel approach to improve processor performance with page replacement techniques" Proceedings of the International Conference on

Information and Communication Technologies, ICICT 2014, 3-5 December 2014

[7] Hasan M H Owda , Munam Ali Shah, Abuelgasim Ibrahim Musa, Manzoor Ilahi Tamimy " A Comparison of Page Replacement Algorithms in Linux Memory Management" International Journal of Computer and Information Technology (ISSN: 2279 – 0764) Volume 03 – Issue 03, May 2014 pp. 565-569

[8] Mr.C.C.Kavar, Mr. S.S.Parmar " Performance Analysis of LRU Page Replacement Algorithm with Reference to different Data Structure" International Journal of Engineering Research and Applications (IJERA) Vol. 3, Issue 1, January -February 2013, pp.2070-2076

★★★