ISSN:0975-9646

Visualizing Page Replacement Techniques based on Page Frequency and Memory Access Pattern

Ruchin Gupta, Narendra Teotia
Information Technology, Ajay Kumar Garg Engineering College
Ghaziabad, India

Abstract— **Virtual memory technique is used by many operating systems. This technique requires an efficient page replacement technique because it severely affects the performance of a computer system. Some of the page replacement techniques are first in first out, least recently used, optimal etc. Optimal has already been proven to be the best. Considerable research has been done to evaluate theses policies and to develop new ones based on recency, frequency, token, and locality model parameters etc. This paper uses a histogram (based on page frequency) and memory access pattern based approach to visualize and to compare least recently used and optimal policies to determine their behaviors. The simulation uses reduced SPEC2000 benchmark traces. Results show that histogram of least recently used and optimal policy equalizes as number of frames increases. Optimal policy histogram equalizes more rapidly than the histogram of least recently used one. Also pages of large frequency of occurrence contribute much to the total number of page faults in both least recently used and optimal page replacement algorithms. It visualizes and concludes that lru follows optimal in memory access pattern for page faults but with an increased number of frames in comparison to optimal.**

Keywords—operating system, lru,page replacement algorithm.

I. Introduction

Operating system uses the concept of virtual memory to provide an illusion to a user having a very large amount of main memory available and allowing him/her to execute a program to be partially there in main memory [1]. In most of the operating systems virtual memory in implemented by demand paging. There are several advantages of using this concept like less I/O, efficient resource utilization etc. Performance of virtual memory is affected by the choice of page replacement technique used. There are several page replacement techniques suggested by different researchers. Elizabeth J. O'Neil1, Patrick E. O'Neil1, Gerhard Weikum uses LRU-K page replacement algorithm or database disk buffering [2]. Sedigheh Khajoueinejad, Mojtaba Sabeghi, Azam Sadeghzadeh used fuzzy cache replacement policy [3]. Optimal technique is proven to be the best but it can not be implemented because it requires future knowledge of the reference string. Least recently used policy approximates it and that is the reason different variations of lru have been implemented. It is well known however that there are many situations where lru behaves far from optimal [4]. Under lru an allocated memory page of a program will become a replacement candidate if the page has not been accessed for a certain period of time under two conditions: (1) the program does not need to access the page; and (2) the program is conducting page faults (a sleeping process) so that it is not able to access the page although it might have done so without the page faults. However, lru page replacement implementations do not discriminate between two types of lru pages and treat them equally [5]. So it means that lru can be made closer to optimal policy by making improvement in to that. Some page replacement policies and some other definitions are given below.

A. *First in, First out (fifo)*

In this when a page is needed, the page that has been in memory for the longest period of time is selected for replacement. The reason is that a page that has only recently been swapped in will have a higher probability of being used again soon.

B. *Least Recently Used (lru)*

Least recently used page replacement policy is based on the assumption that the page reference pattern in the recent past is a mirror of the pattern in the near future. Pages that have been accessed recently are likely to continue to be accessed and ought to be kept in physical memory.

C. *Optimal method*

This policy selects a page for replacement which will be used after the longest period of time. Since this requires future knowledge of the reference string, this can not be implemented in the system. Hence this policy is used for comparative study only.

D. *Memory Access Pattern*

The memory access pattern shows how different pages are used as the application executes. X axis shows the progress of time while the Y axis shows which page numbers are used at different times during the execution of an application. It shows the memory access pattern of an application during execution of that particular application.

E. *Histogram*

The histogram is very popular in digital imaging and other subjects. A histogram for any given data set shows the frequency of occurrence of every element in the given data set. It gives a clear idea about the high frequency element and low and medium frequency elements for a given data set. It is assumed that data set contains similar kind of data in the given data set.

II. Simulator

Here we develop a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table in the single programming model. The simulator keeps track of what pages are loaded into memory. As it processes each memory event from the trace, it should check to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. Here we take all pages and page frames of 4 KB size.

We used trace-driven simulations to obtain results. The traces were generated with the SimpleScalar toolset [6] using a simple in-order processor model. In all experiments, the page size was set to 4KB, which is the page size used in the Simple Scalar simulators.

The set of benchmarks used is a subset of the SPEC 2000 benchmark suite. We used reduced input data sets and such reduced inputs were not provided for all benchmarks. In order to reduce the simulation time, we used the large reduced input datasets described in [7]. The execution characteristics of these reduced input datasets are similar to the execution profiles of the SPEC 2000 reference datasets.

It implements 2 page replacement algorithms least recently used and optimal. The simulator is written in C in MS-DOS environment.

Numbers of frames are varied and no of page faults are calculated. Different traces are taken as inputs and numbers of page faults are calculated. Also MATLAB is used to plot memory access pattern plot for different traces and to plot histograms for page fault behavior of different traces.

Each trace obtained from the SPEC2000 benchmark is a real recording of a running program. Real traces are enormously big having billions and billions of memory accesses. However, a relatively small trace is enough. Each trace only consists of one million memory accesses. Traces are gcc.trace.gz, swim.trace.gz , bzip.trace.gz.

Each trace is a series of lines, each listing hexadecimal memory addresses followed by R or W to indicate a read or a write. For example, gcc.trace trace starts like this:

0041f7a0 R 13f5e2c0 R 05e78900 R 004758a0 R

III. Simulation Results

Applications swim, gcc, bzip show different memory access patterns. Swim application shows quite a stable pattern of memory access. Also different applications have different number of pages. Bzip application exhibits more correlated access than swim and gcc application. Figure 1, 2, 3 shows the memory access pattern for reduced swim, gcc, bzip traces.

Figure 4, 5, 6 shows histograms for 3 applications swim, gcc, and bzip respectively. From figure 7 to 24 it shows the histograms for page faults for different traces using different number of frames using lru and optimal. From figure 25 to 42 it shows the memory access patterns for page faults of different traces using lru ,optimal with different number of frames. From figure 25 to 40 each vertical line shows a page fault and gap or valley indicate no page fault. X axis here shows the refernce time while y axis shows the page number.

From figure 4, 5, 6 it is clear that a few pages are used very heavily while few pages are used very rarely .It is clear from figure 10 page numbers near 100000 have very high frequency of occurrence .Also these page numbers are being used quite stably over a uniform pattern.

Also from figure 7, 8, 9, it is observed hat as no. of frames increase for lru, page faults occur on same pages but with less frequency .Also as no of frames increases, lru equalizes no of page faults for different pages.

From figure 10, 11, 12 (histogram for page faults for optimal), It is clear that optimal also equalizes the no of page faults as no of frames increases but more than lru.

For lru in figure 7, the highest no of page faults are not generated for the highest frequency of occurrence page numbers. Also from figure 7, 8, 9 it is concluded that the page numbers on which more no. of page faults occur change with the increase in no. of frames.

From figure 13, 14, 15,16,17,18 it is clear that both lru and optimal generate largest no of page faults for pages not having the highest frequency of occurrence. Also in both, as no of frames increases the pages on which largest no of page faults occur remain almost same.

From figure 19, 20, 21, it is clear that for 5 frames lru generates largest no of page faults for pages of the highest frequency of occurrence. Also for lru, as no of frames increases, the pages on which more no. of page faults occur change. The same is true for optimal policy in figure 22, 23, 24. Figure 25 to 42 show that lru approximates optimal by generating large number of page faults common to optimal. For swim trace; memory access patterns of lru and optimal are quite close as number of frames increases.

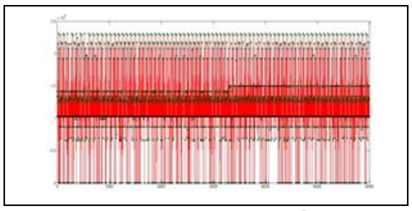
Fig. 1. swim application

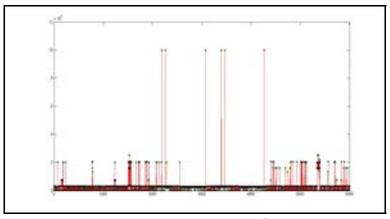
Fig. 2. gcc application

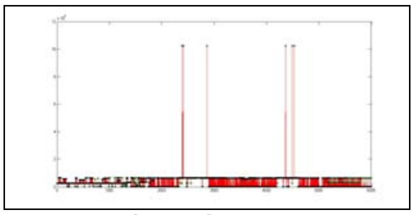
Fig. 3. bzip applicatipon

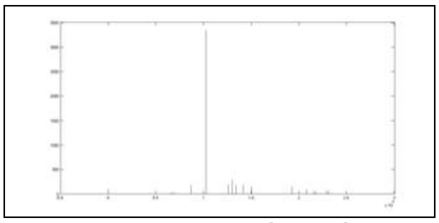
Fig. 4. swim application histogram

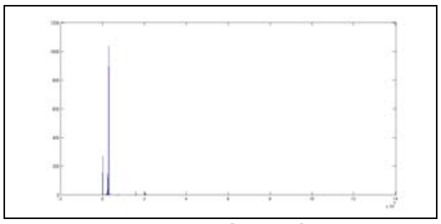
Fig. 5. gcc application histogram

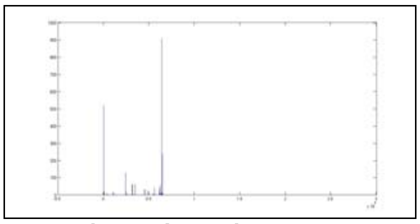
Fig. 6. bzip application histogram

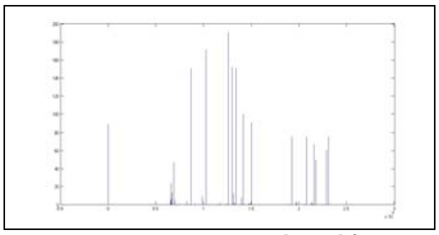
Fig. 7. Swim using lru and frames =5

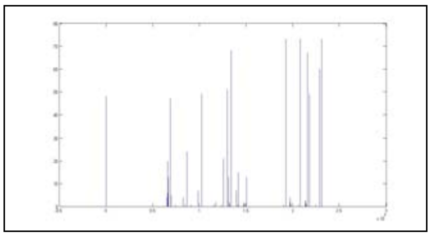
Fig. 8. Swim using lru and frames =15

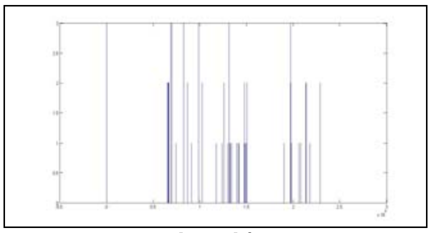
Fig. 9. Swim using lru and frames =5 0

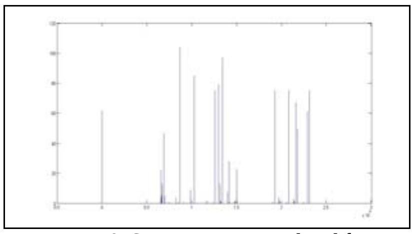
Fig. 10. Swim using optimal and frames =5

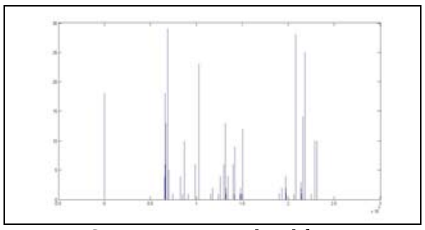
Fig. 11. Swim using optimal and frames =15

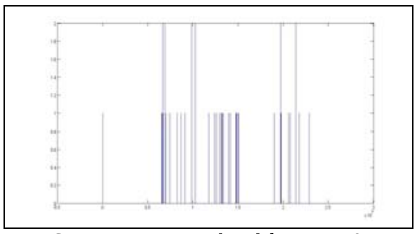
Fig. 12. Swim using optimal and frames =50

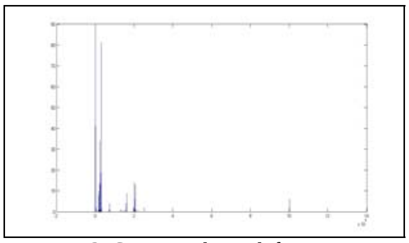
Fig. 13. Gcc using lru and frames =5

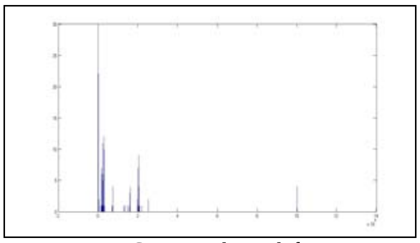
Fig. 14. Gcc using lru and frames =15

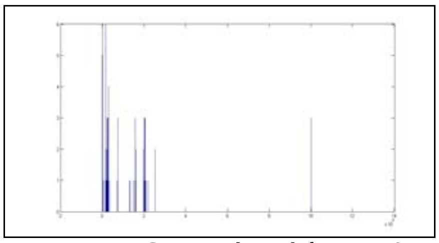
Fig. 15. Gcc using lru and frames =50

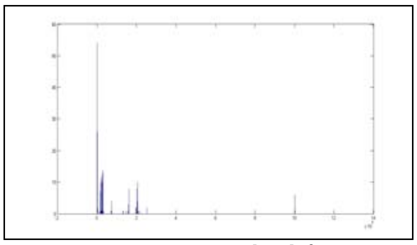
Fig. 16. Gcc using optimal and frames =5

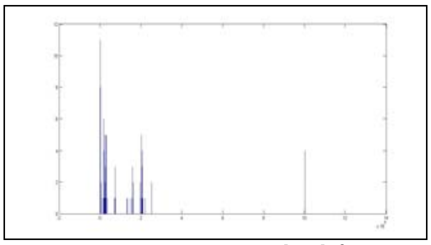
Fig. 17. Gcc using optimal and frames =15

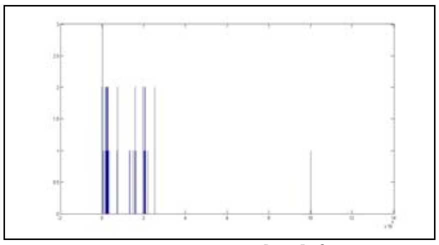
Fig. 18. Gcc using optimal and frames =50

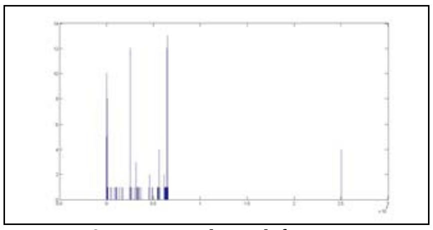
Fig. 19. Bzip using lru and frames =5

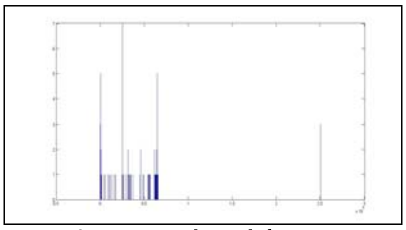
Fig. 20. Bzip using lru and frames =15

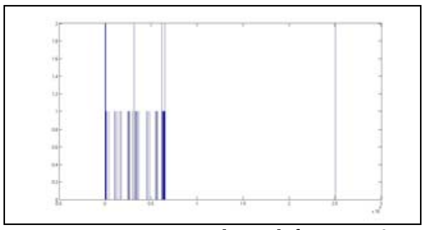
Fig. 21. Bzip using lru and frames =50

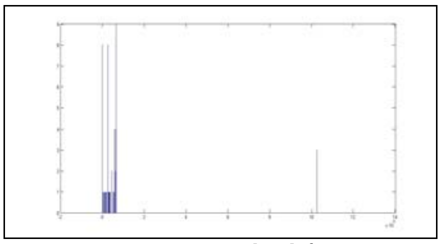
Fig. 22. Bzip using optimal and frames =5

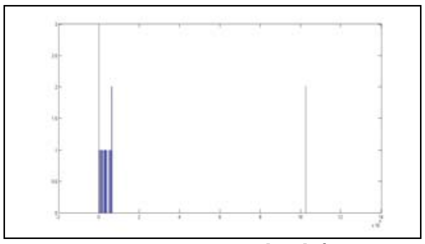
Fig. 23. Bzip using optimal and frames =15

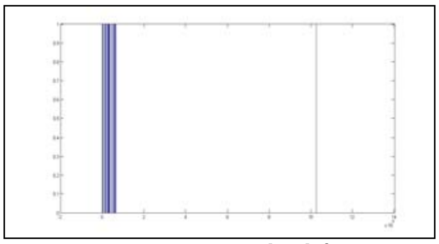
Fig. 24. Bzip using optimal and frames =50

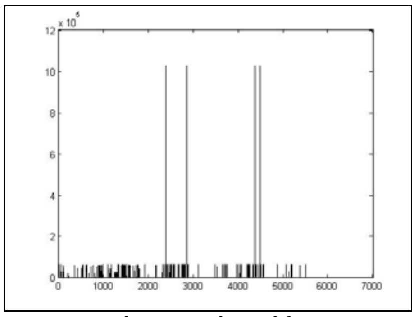

Fig. 25. bzip using lru and frames=5

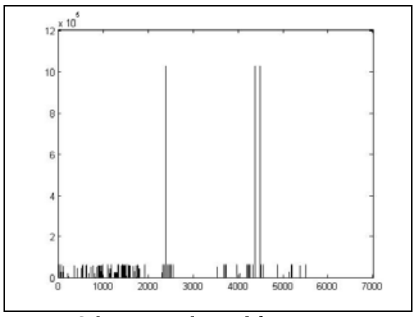

Fig. 26 bzip using lru and frames =15

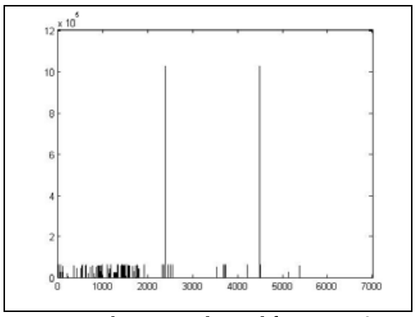

Fig. 27. bzip using lru and frames = 50

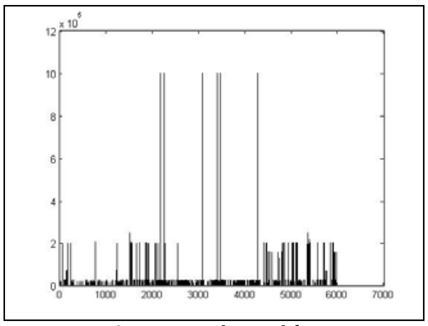

Fig. 28. gcc using lru and frames= 5

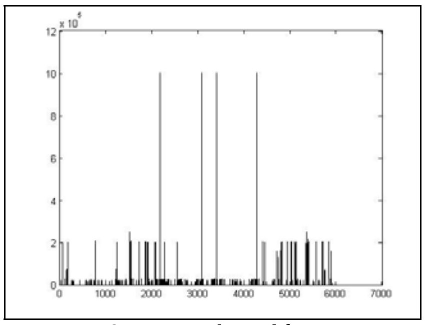

Fig. 29. gcc using lru and frames = 15

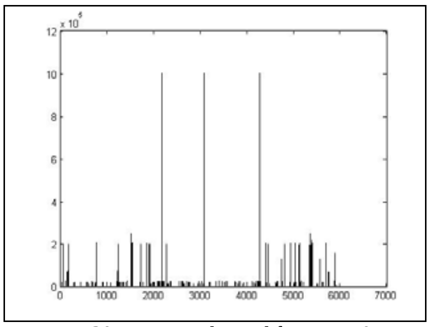

Fig. 30. gcc using lru and frames =50

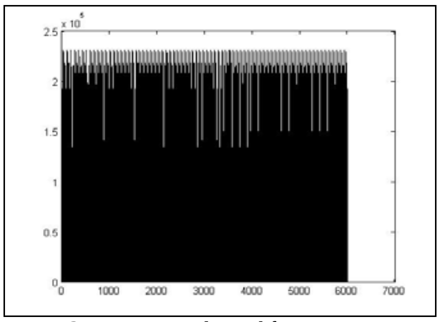

Fig. 31. swim using lru and frames =5

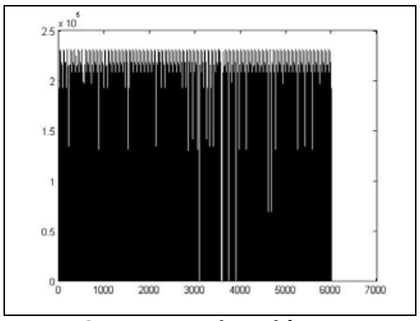

Fig. 32. swim using lru and frames =15

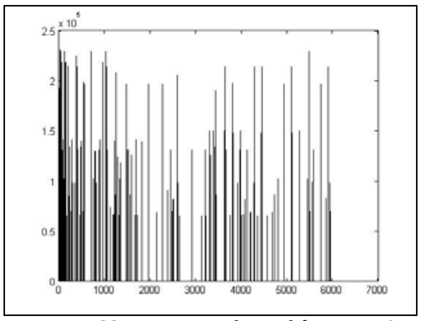

Fig. 33. swim using lru and frames =50

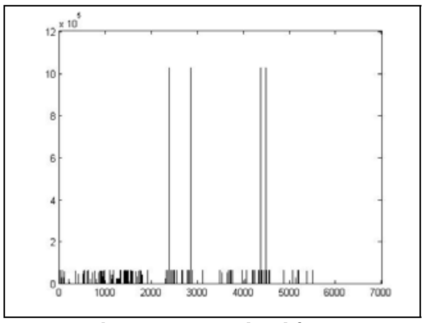

Fig. 34. bzip uing optimal and frames =5

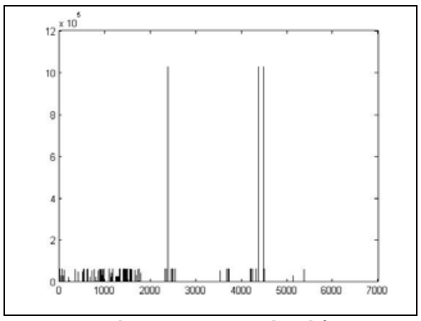

Fig. 35. bzip using optimal and frames =15

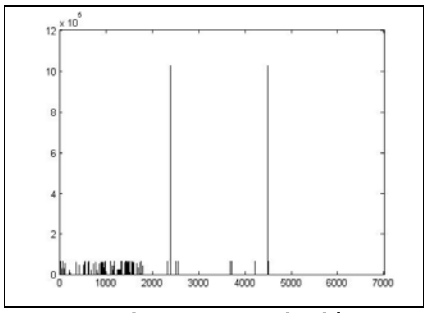

Fig. 36. bzip using optimal and frames =50

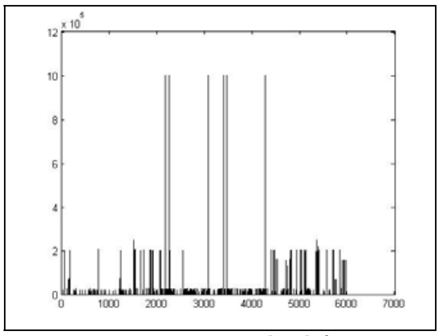

Fig. 37. gcc using optimal and frames =5

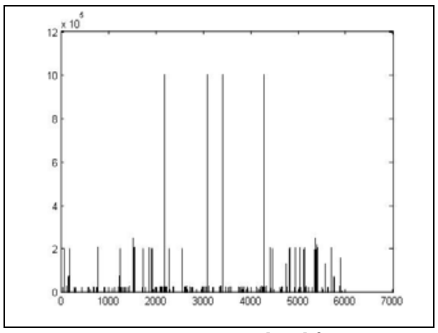

Fig. 38.gcc using optimal and frames =15

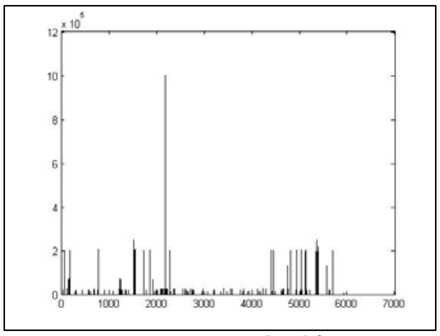

Fig. 39. gcc using optimal and frames =50

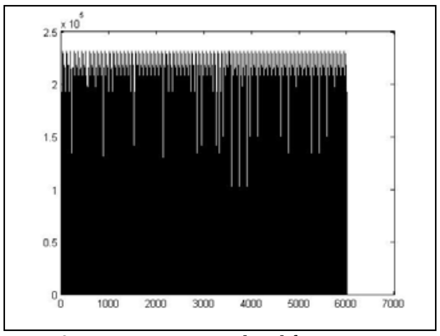
Fig. 40. swim using optimal and frames =5

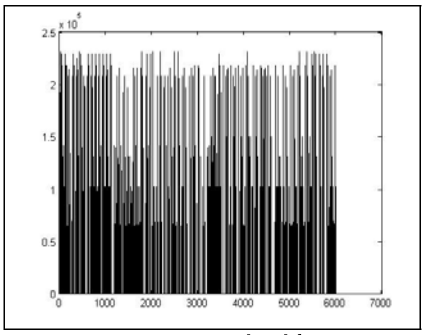
Fig. 41. swim using optimal and frames =15

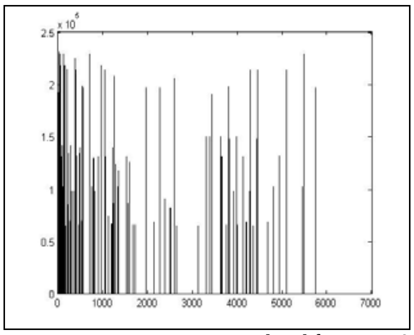
Fig. 42. swim using optimal and frames =50

CONCLUSIONS

It is observed that optimal and LRU both may and may not generate largest no of page faults on page numbers of the highest frequency and it depends upon memory access pattern. It is found that as number of frames increases, both lru, optimal generates page faults on almost the same page numbers with less number of page faults. For both lru and optimal; the number of page faults on pages of high frequency of occurrence change with no. of frames.

It concludes that pages of high frequency of occurrence contribute much to the total number of page faults for lru policy. Also high frequency pages contribution to the total number of page faults varies with the number of frames present in main memory. The same applies to the optimal policy.

It is found that pages of high frequency of occurrence contribute a good amount to the total no. of number of page faults for both lru and optimal policies. Also in some cases, contribution of high frequency pages to total no of page faults is less in optimal than lru policy.

It is also observed that histograms for both lru and optimal policies equalize as the number of frames increases. Also histogram for optimal policy equalizes more rapidly than lru. It is also observed that lru well approximates optimal. Lru generates a large number of same page faults (as in optimal) but with an increased number of frames than compared to optimal.

To the best of authors' knowledge; no such kind of work has been reported in the literature hence comparative study to the previous work is not possible.

REFERENCES

[1] Abraham Silberschatz, Peter Baer, 1999, Operating System Concepts (5th Ed.).New York: John Wiley & Sons, Inc.

[2] Elizabeth J. O'Neil1, Patrick E. O'Neil1, Gerhard Weikum, the LRU-K Page Replacement Algorithm For Database Disk Buffering, SIGMOD Washington, DC, USA 1993 ACM.

[3] Sedigheh Khajoueinejad, Mojtaba Sabeghi, Azam Sadeghzadeh, A Fuzzy Cache Replacement Policy and its Experimental Performance Assessment, 2006 IEEE.

[4] Ben Juurlink, Approximating the Optimal Replacement Algorithm, *CF'04,* April 14–16, 2004, ACM 1581137419/ 04/0004.

[5] Song Jianga,, Xiaodong Zhangb,, Token-ordered LRU: an effective page replacement policy and its implementation in Linux systems, 2004 Elsevier .

[6] D. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, Univ. of Wisconsin-Madison, Comp. Sci. Dept., 1997.

[7] AJ KleinOsowski, John Flynn, Nancy Meares, and David J. Lilja. Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research. In Proc. Workshop on Workload Characterization, Int. Conf. on Computer Design (ICCD), 2000.