J. Basic. Appl. Sci. Res., 2(10)10377-10383, 2012

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Presenting a Novel Page Replacement Algorithm Based on LRU

Ali Khosrozadeh, Sanaz Pashmforoush, Abolfazl Akbari, Maryam Bagheri, Neda Beikmahdavi

Department of Computer Engineering, Ayatollah Amoli Branch, Islamic Azad University, Amol, Iran

ABSTRACT

In this paper we are going to reviewing the page replacement algorithms and comparing the performance of various algorithms in this field. We succeed that represent a new page replacement algorithm according to LRU method that has a better performance than former methods. Our major attempt was to substitute some former method characteristic based on new idea.in this competition we evaluate new policy that precisely, needed some components for select page for replacement. But key concept is the replacement algorithm must impose no overhead to system.
Keywords: Page Replacement, LRU Algorithm, Novel Page Replacement Algorithm

1. INTRODUCTION

Managing the memory in a computer system requires two or more levels of memory to be controlled. Among these levels, one with shortest access time is selected as the first level, also called primary memory. Because of short access time and so, the fast access, such this memory is expensive while having lower storage space. This limited space, in turn, yields to fast overflow and so, if more memory is required, some parts of this memory must be empty.

A major method to managing the memory space and distributing the limited space among the applications to be executed was the memory segmentation. But, because of external fragmentation phenomenon which yielded to apparently wasting the memory space, this method was replaced with the paging method. Also, the latent memory wasting was the reason for the paging method to be inefficient. So a method based on a combination of these two models was introduced to resolve their problems. In a paging basis, the available memory space is divided into blocks known as page frames. Each application asking the memory is divided into some pages and will be given a number of page frames to contain some of these pages. This number is determined by the operating system and can be different for applications. Until the memory contains at least one empty page frame, designating the frames for the applications and loading the pages into them is easy, relatively.

But, the main problem in managing the frames will be evolved when all available frames have been designated to the applications and there are no empty frames. The solution for this, will be expelling one or more pages from the memory and free some frames to be designated for new pages. It is important to determine what page(s) must be expelled. Removing a page that may be required in close future can affect the system performance; because imposes a reloading time overhead.

These challenges have motivated the researchers to introduce page replacement algorithms, which select a page to be replaced with a new one, based on determined criterions, politics and purposes. In diverse conditions, diverse algorithms have different performances. Among the proposed methods, FIFO, NRU, Second Chance, Clock, LRU and Optimal algorithms can be mentioned. Also, among these, the optimal algorithm has the best performance; but it has been impractical already and only can be used as a comparison reference to evaluate other methods. After this algorithm, Least Frequently Used (LRU) has been known as the most efficient algorithm. This algorithm considers the number of times which each resident page in the memory has been referred to, and selects a page with minimum counter value to be moved out of memory. Since this algorithm has been the most efficient method among the practical algorithms, most of researches have focused on it to improve the performance. Also, this has created a motivation to present a method based on introducing an extra parameter aside the parameter used by LRU, such that the page replacement process can be performed more efficiently.

So, for the purpose of this paper, it has been organized as follows: in section 2, some page replacement algorithms, previously presented, include FIFO, NRU, Second Chance, Clock, LRU and Optimal will be reviewed. The proposed algorithm will be presented in section 3 and its criterion parameters will be introduced there. Section 4 has been designated for reporting the experiments results and comparisons between the mentioned method performances. Finally, section 5 will contain the conclusions.

2. The Page Replacement Algorithms

As mentioned earlier, during a program execution, usually, some data are required to be loaded into the primary memory. In the memory management, based on paging mechanism, the required data are entered to the memory as pages contents. When a required page is not found in memory, a page fault event is occurred. In these cases, the required page must be loaded into memory. But if all page frames are full, the main problem will be finding an

***Corresponding Author:** Ali Khosrozadeh, Department of Computer Engineering, Ayatollah Amoli Branch, Islamic Azad University, Amol, Iran, a.khosrozadeh@iauamol.ac.ir

appropriate frame to contain the required page. Really, the operating system must select a page frame in primary memory and move its content out of it so that an empty page frame is provided for the new page, entering to the memory [2]. Although, when a page fault occurs, one frame can be selected randomly, in order to increase the system performance, it will be better to select a page that don't affect the system and program execution politics; because, if a page with high usage ratio is selected and move out of primary memory, it may be required to be reloaded, soon. When a page contents are loaded into primary memory, actually, they are read from a secondary memory (input operation). Also, when a page is moved out of primary memory, it must be written on a secondary memory (output operation). So, a page fault occurring can impose Input/Output operations to the system. As can be seen in Fig. 1, reducing I/O in multi programming models increases the CPU performance [4].

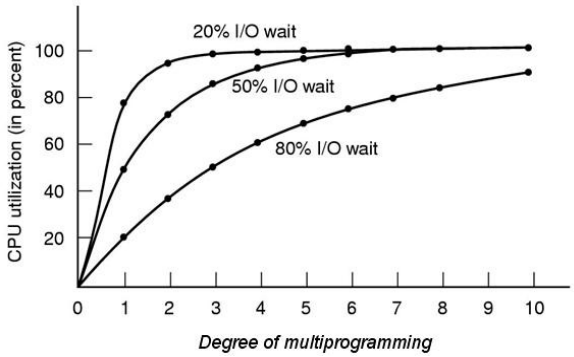

Fig. 1: Increasing CPU performance by I/O reducing

Appropriate selecting the pages can reduce the page fault occurs and so, can reduce the I/O which in turn, increases the system performance. Since now, many studies about page replacement algorithms are done and several algorithms have been presented practically and theoretically. In the following, some of most important ones will be investigated.

2.1. The First-In-First-Out (FIFO)

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 [5]. 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 [8].

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.

Fig. 2 and fig. 3 depict the FIFO execution for different frames number. Fig. 3 shows that in special sequences, increasing frames number yields to page fault ratio increasing. So, this algorithm performance reduces regarding frame number growing.

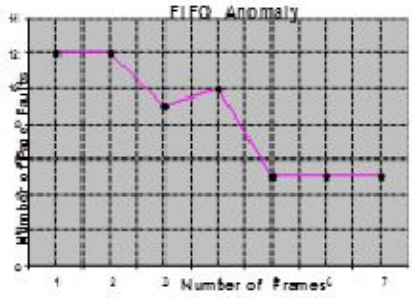

Fig. 2: The FIFO performance difference for various frames number

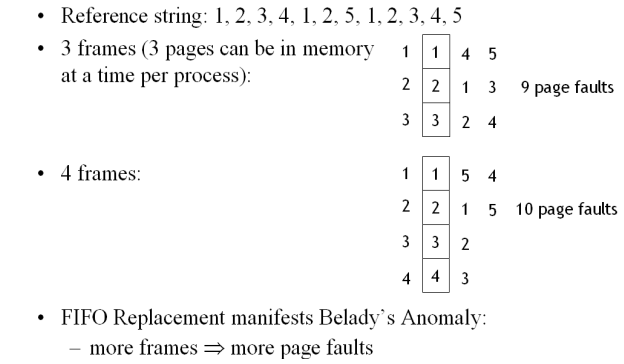

Fig. 3: FIFO anomaly diagram

2.2. Not Recently Used (NRU)

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 reference bits resetting for pages. This phenomenon leads to have pages that are modified while not be referenced.

2.3. The Second Chance Algorithm

Applying a simple change in FIFO can prevent from removing pages that are used widely. In the modified FIFO, named the Second Chance algorithm, a reference bit (R) with a definition similar to above mentioned bit (for NRU) is considered along with the pages histories. This means that, for selecting a page to be moved out, the reference bit for the oldest resident page is investigated. If it's zero, the intended page is known as both an old and useless page, and so, it will be replaced with a new page. But if R=1 for the oldest page, it will be cleared and the associated page will be moved to the end of queue, similar to a page that has been arrived to the memory, just now [9]. Then the search will continue. Really, this algorithm looks for a page which has not been referred to in previous clock pulses [7]. If all pages have R=1, this algorithm operates similar to FIFO. Fig.4 shows how this algorithm operates.

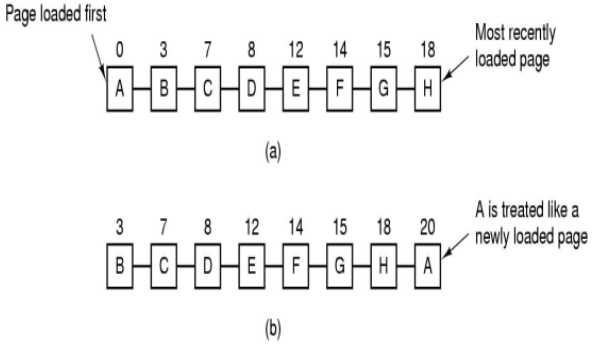

Fig. 4: The second chance algorithm operation

2.4. 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 fig. 5, 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 [1].

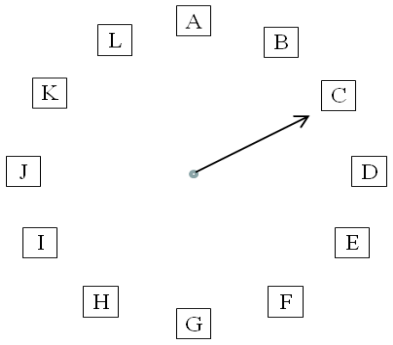

Fig. 5: The clock algorithm schematic form

2.5. Least Recently Used (LRU)

The idea behind this algorithm is based on this fact that the pages widely used in the last few instructions, will probably be used heavily again, in the next few ones. And, in contrast, the pages that have not been used for long times, will probably remain unused afterward. So, this algorithm selects a page that its last usage is before all other resident pages. This idea is based on Locality theory [2]. The LRU algorithm has been used as a common page replacement algorithm and has been considered as a successful algorithm. Fig. 6 shows this algorithm functionality by considering a random page calling sequence.

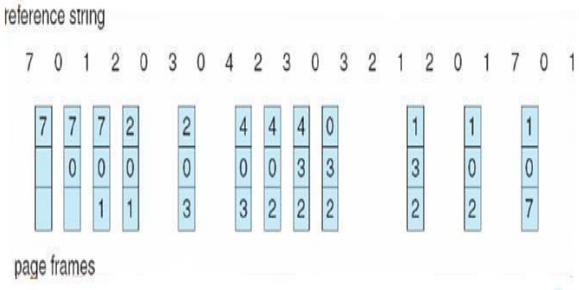

Fig. 6: LRU algorithm execution by considering 3 page frames

Although this algorithm has a suitable performance, as a disadvantage, its difficult and expensive implementation can be addressed [3]. Fully implementing LRU requires maintaining a linked list of all pages in memory, in which, the most recently used page is placed at the front, and the least recently used page is placed at the rear. This list must be updated on every memory reference. This includes finding a page in the list, deleting it, and then moving it to the front. But it is a very time consuming operation, even in hardware (assuming that such hardware could be built). As another ways to implement LRU, two methods can be considered:

- Using a stack – In this case, each invoked page is inserted on the top of stack. So the pages that have most usage, recently, are put on the stack top, always. Fig. 7 shows how this stack works [6].

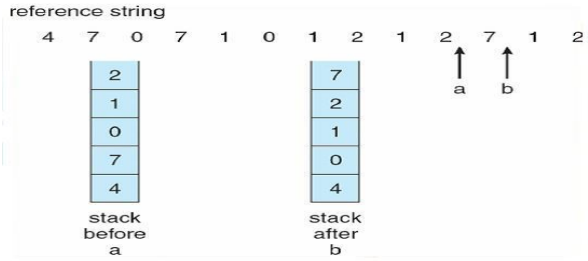

Fig. 7: The hardware Implementation for LRU by the stack

- Using a counter – In this method, a 64-bit counter (C) is used. After executing each instruction, this counter will be incremented, automatically. The page table entry for each resident page includes a field that can store the counter value, and when a page is referred to, the current value of C is stored in its page table entry. When a page fault occurs, all counter values stored in page table entries are examined and the lowest one is selected. This minimum value corresponds to a page that has been least recently used. So this page will be moved out of memory and its location is designated for a new page.

2.6. The Optimal Algorithm

Among all page replacement algorithms, the optimal algorithm has the best performance; but its implementation has been impossible, since now. In this algorithm, when a page fault occurs, if there is any empty frame, it will contain the new invoked page. But the main problem will be selecting a frame to be vacated if there is no empty frame. To solve this problem, a number is assigned to each page in the memory. This numeric tag specifies the number of operations that must be performed before referring to it's correspond page. According to this, it must be clear that a page with maximum tag value will go out of memory. In the other word, the operating system tries to avoid undesirable events as much as possible. Or simply, one page is selected to be moved out which its first following usage is expected to be after all other pages usages.

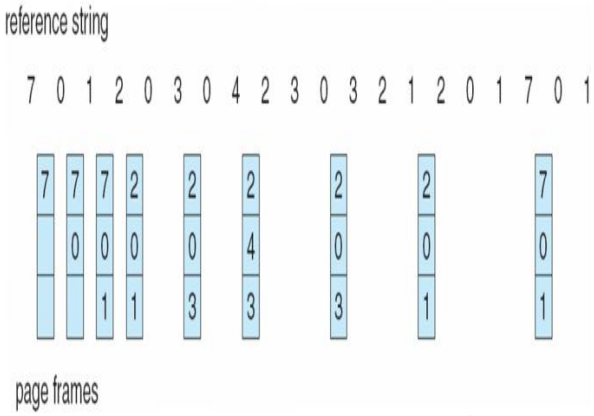

Fig. 8: Optimal algorithm execution with 3 frames

However, one of reasons for designing this algorithm, while it is not practically applicable, is its usage as an ideal reference for evaluating and comparing practical (and not theoretical) algorithms. Fig. 8 shows this algorithm execution by considering a random page invoking sequence. As previously mentioned, among all practical algorithms for page replacement in primary memory, the LRU algorithm has been known as the best performance one. But this algorithm, via its challenges, has been studied and considered for presenting new algorithms with better performances. As explained in section 2, the LRU algorithm uses for each page, the number of times that the page has been referred, recently. This paper, at the following, will present a method that uses a new parameter along with the LRU parameter, to select the page that must be sent out of memory.

3. Presenting the Proposed Method

In this section, an algorithm based on LRU will be introduced, and will be referred to as PRO_LRU, here. The first and most important note about this algorithm is using an extra feature for selecting the outgoing pages, together with the feature, used by LRU. While this secondary feature interacts with the first one (the LRU parameter), is assumed to has a more determining role, among two features. In the other word, the proposed algorithm has two indicators for selecting the best page:

1. The total number of references, to now (TNR)
2. The spent time since last reference (STR)

In this algorithm, when a page fault occurs, the first parameter, TNR, will be investigated for all pages. If there is only one page with minimum TNR value, this page will be selected to exit the memory. And so, in such these cases, the algorithm operates similar to LRU. Otherwise, if the minimum TNR value is shared between special fractions of pages (r), the second parameter comes to be mentioned. In the other word, when the number of pages with minimum TNR value is more than one, the second parameter (STR) will be considered to selecting the page, among this set of filtered pages. In this case, considering all pages with minimum TNR value, a page with most STR value will be selected, that is, a page that most time has been spent from its last reference. Fig. 9 shows a flowchart describing this process. Also, Fig. 10, shows the PRO_LRU for 3 page frames, considering the rate r=2/3.

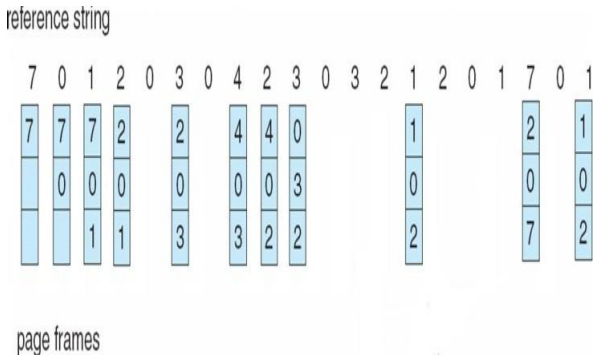

Fig. 10: Executing PRO_LRU for 3-frame memory

3. EXPERIMENTAL RESULTS

In order to evaluate the proposed algorithm and order to evaluate the proposed algorithm and comparing it with the other algorithms, they have been simulated using MATLAB. Some experiments have been done and in each, a number of page invoking sequences have been used to computing the performance. In the first one, the Hit ratio has been considered as the compare criterion. For this purpose, 100 random sequences with length 25 were imposed to LRU and PRO-LRU, as page invoking sequences. In this experiment, four cases for memory frame number include 2-Frame, 3-Frame, 4-Frame and 5-Frame were considered and the results have been reported in Table (1). Based on this table, the bar diagram for these results can be shown as fig. 11.

Table 1: Hit ratio values for LRU and PRO-LRU regarding diverse memory sizes

The Model	Hit Ratio (%)			
	2 FRAME	3 FRAME	4 FRAME	5 FRAME
PRO-LRU	13.96	20.91	26.66	31.37
LRU	13.75	18.87	23.71	25.63

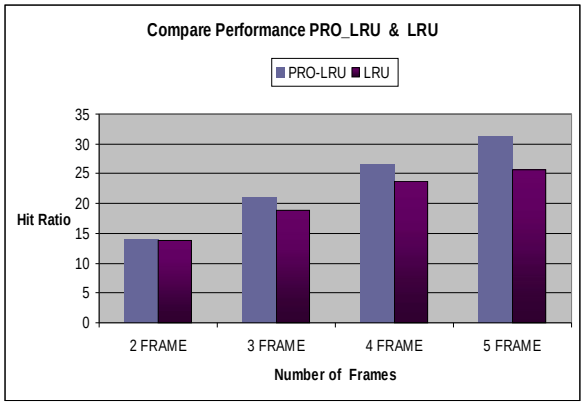

Fig. 11: Comparing LRU and PRO-LRU for several memory frame numbers, regarding Hit ratio

In the second experiment, the Hit ratio has been considered as the compare criterion, again. But in this experiment, the difference between Hit ratio values, per diverse memory sizes (in term of frame number) is considered. For this purpose, by considering each memory size, 100 random sequences with length 25 were imposed to LRU and PRO-LRU. For each size, the rate of invoking per which the Hit ratio for PRO-LRU is lower than LRU Hit ratio, is reported. The same is repeated for PRO-LRU=LRU and PRO-LRU>LRU. In this experiment, four cases for memory size include 2-Frame, 3-Frame, 4-Frame and 5-Frame were considered, and the results have been reported in Table (2). Also, based on this table, the bar diagram for these results can be shown as fig. 12.

Table 2: The results of the second experiment: Comparing Hit ratios differences

Memory Size	Failure PRO-LRU < LRU	Equal PRO-LRU = LRU	Success PRO-LRU > LRU
2 Frame	33%	38.50%	28%
3 Frame	22.80%	24.50%	52.60%
4 Frame	17.50%	21.05%	61.40%
5 Frame	14.03%	17.50%	68.42%

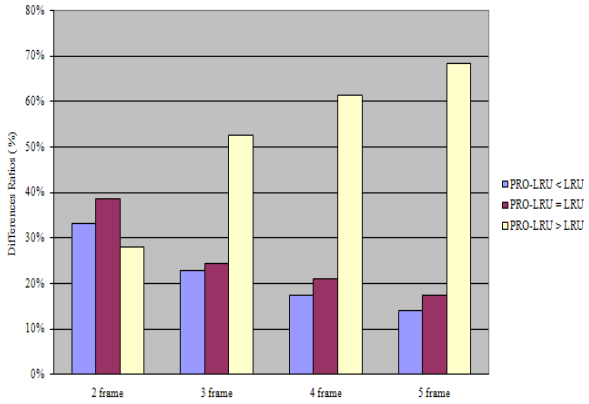

Fig. 12: The bar diagram for comparing Hit ratios differences

5. Conclusion

Studying the page replacement algorithms has indicated that among all practical algorithms, the LRU has had the best performance, and the next researches have focused on this algorithm, for improving the performance. In this paper, a page replacement algorithm based on LRU was presented. This algorithm used a complementary parameter beside the LRU parameter, to determining the page that must be replaced with a new one. Considering the results of experiments shows that adding this parameter can improve the performance of page replacement process. Investigating above tables and figures indicates that this parameter (page reference number) can offer better improvements while the number of frames increases.

6. Acknowledgment

I would like to thanks the Islamic Azad University- Ayatollah Amoli Branch, for financial support of this research project.

7. REFERENCES

[1] Brotherton, J., Torng, E., "Victim Caching with Better Block Replacement", Department of Computer Science, Michigan State University, pp. 162 – 182, 1996.

[2] O'Neil, J. E., O'Neil, E. P., Weikum, G., "The LRU-K Page Replacement Algorithm for Database Disk Buffering", In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 297 – 306, 1993.

[3] O'Neil, J. E., O'Neil, E. P., Weikum, G., "An Optimality Proof of the LRU-K Page Replacement Algorithm", Journal of the ACM, Vol. 46, No. 1, pp. 92 – 112, January 1999.

[4] Ghasemzadeh, H., Mazrouee, S., Kakoee, M. R., "Modified pseudo LRU Replacement Algorithm", 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, pp. 137 – 145, 2006.

[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] Kampe, M., Stenstrom, P., Dubois, M., "Self-Correcting LRU Replacement Policies", Department of Electrical Engineering – Systems University of Southern California, pp. 253 – 261, 2002.

[7] Kedzierski, K., Moreto, M., Cazorla, F., J., Valero, M., "Adapting Cache Partitioning Algorithms to pseudo-LRU Replacement Policies", Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on Issue Date: 19 - 23 April 2010, pp. 1 – 12, 2010.

[8] Kim, K., Park, K., "Least Popularity – Per – Byte Replacement Algorithm for a Proxy Cache ", IEEE, pp. 780 – 787, 2001.

[9] Khuri, S., Hsu, H. C., "Visualizing the CPU Scheduler and Page Replacement Algorithms", Journal of the ACM, pp. 227 – 231, 1999.