International Journal of Electronics, Communication & Soft Computing Science and Engineering
ISSN: 2277-9477, Volume 2, Issue 4

Page Replacement Algorithms – An Evolution

V. M. Aswar **Prof. A. P. Bodkhe**

***Abstract* —The operating system uses virtual memory for better memory utilization. A virtual memory management system needs efficient page replacement algorithms to decide which pages to evict from memory in case of a page fault. Over the years many algorithms have been proposed for page replacement. Each algorithm attempts to minimize the page fault rate while incurring minimum overhead. As newer memory access patterns were explored, research mainly focused on formulating newer approaches to page replacement which could adapt to changing workloads. This paper attempts to summarize major page replacement algorithms proposed till date. We look at the traditional algorithms such as LRU and CLOCK, and also study the recent approaches such as LIRS, CLOCK-Pro, ARC, and CAR.**

***Key Words* — Page Replacement, LRU, LIRS, CLOCK-Pro, ARC, CAR.**

I. INTRODUCTION

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.

In order to realize the full potential of multiprogramming systems it is essential to interleave the execution of more programs than can be physically accommodated in main memory. Hence we use a two – level memory hierarchy consisting of a faster but costlier main memory and a slower but cheaper secondary memory.

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 [7]. Pages are brought into main memory only when the executing process demands them; this is known as demand paging.

A page fault is said to occur when a requested page is not in main memory and needs to be brought from secondary memory. In such a case an existing page needs to be discarded. The selection of such a page is performed by page replacement algorithms which try to minimize the page fault rate at the least overhead.

This paper outlines the major advanced page replacement algorithms in chronological order along with their relative pros and cons. Here we have discussed basic algorithms such as Belady's MIN, LRU and CLOCK and move on to the more advanced Dueling CLOCK, LRU-K, LIRS, CLOCK-Pro, ARC and CAR algorithms.

II. PAGE REPLACEMENT ALGORITHMS

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed has been modified, while in memory, it must be rewritten to the disk to bring the disk copy up to date. If, however, the page has not been changed (e.g., it contains program text), the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted.

While it would be possible to pick a random page to evict at each page fault, system performance is much better if a page that is not heavily used is chosen. If a heavily used page is removed, it will probably have to be brought back in quickly, resulting in extra overhead. Much work has been done on the subject of page replacement algorithms, both theoretical and experimental.

The best page replacement algorithm is clearly the one that manages the pages of a running process in such a way, that the execution of the process produces the least possible number of page faults. This way the running process does not wait for pages being brought back from the swap area after a page fault. Therefore the process completes sooner.[10]

A. First In First Out (FIFO)

The simplest page-replacement algorithm is a FIFO algorithm. A FIFO 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. Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.[9]

B. Belady's MIN

This is the most optimal algorithm which guarantees best performance. It suggests removing the page from the memory which is not to be accessed for the longest time. This optimal result is referred to as Belady's MIN algorithm or the clairvoyant algorithm. The replacement decisions rely

on knowledge of the future page sequence but it is impossible to predict how far in the future pages will be needed. This makes the algorithm impractical for real systems. This algorithm can be used to compare the effectiveness of other replacement algorithm making it useful in simulation studies since it provides a lower bound on page fault rates under various operating conditions.

C. Least Recently Used (LRU)

If we use the recent past as an approximation of the near future, then we will replace the page that *has not been used* for the longest period of time This approach is the L**east-Recently-Used (LRU)** algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.[8]

The LRU policy is based on the principle of locality which states that program and data references within a process tend to cluster. For a long time, LRU was considered to be the most optimum online policy. The problem with this approach is the difficulty in implementation. One approach would be to tag each page with the time of its last reference; this would have to be done at each memory reference, both instruction and data. LRU policy does nearly as well as an optimal policy, but it is difficult to implement and imposes significant overhead.

D. CLOCK

In CLOCK, all page frames are visualized to be arranged in form of a circular list that resembles a clock.

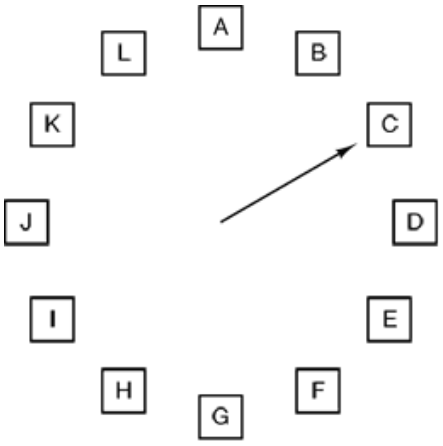


Figure 2.1: The clock page replacement algorithm.[8]

The hand of the clock is used to point to the oldest page in the buffer as shown in Figure 2.1. Each page has an associated reference bit that is set whenever the particular page is referenced. The page replacement policy is invoked in case of a page miss, in which case the page pointed to by the hand, i.e. the oldest page is inspected. If the reference bit of the page is set, then the bit is reset to zero and the hand is advanced to point to the next oldest page. This process continues till a page with reference bit zero is found.

E. Dueling CLOCK

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 is presented and a set dueling approach is used to adaptively choose either the CLOCK or the scan resistant CLOCK as the dominant algorithm for page replacement. In order to make clock scan resistant, the only change required is that the hand of the clock should now point to the newest page in the buffer rather than the oldest page. Now, during a sequential scan, the same page frame is inspected first every time, and only a single page frame is utilized for all pages in the scan keeping the rest of the frequently accessed pages secure in the buffer. The Dueling CLOCK algorithm suggests a method to adaptively use the above two algorithms, namely, CLOCK and scan resistant CLOCK. The cache is divided into three groups, G1, G2, and G3. Small group G1 always uses CLOCK algorithm for replacement while small group G2always uses the scan resistant CLOCK policy. The larger group G3 can use either CLOCK or scan resistant CLOCK policies depending upon the relative performance of groups G1 and G2.

In order to determine the replacement policy for G3, a 10 bit policy select counter (PSEL) is used. The PSEL counter is decremented whenever a cache miss occurs on the cache set in group G1 and the counter is incremented whenever a cache miss occurs on the cache set in group G2.

Now, group G3 adopts CLOCK replacement policy when the MSB of PSEL is 1, other G3 uses the scan resistant CLOCK policy. In practice, Dueling CLOCK algorithm has been shown to provide considerable performance improvement over LRU when applied to the problem of L2 cache management.

F. LRU-K

The LRU policy takes into account the recency information while evicting pages, without considering the

frequency. To consider the frequency information, LRU-K was proposed which evicts pages with the largest backward K-distance. Backward K-distance of a page p is the distance backward from the current time to the K^{th} most recent reference to the page p. Since this policy considers K^{th} most recent reference to a page, it favors pages which are accessed frequently within a short time. Experimental results indicate that LRU-K performs better than LRU [6]; while higher K does not result in an appreciable increase in the performance, but has high implementation overhead.

G. 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].

Figure 2.2 describes a scenario where stack S holds three kinds of blocks: LIR blocks, resident HIR blocks, non-resident HIR blocks, and a list Q holds all of the resident HIR blocks. Each HIR block could either be in stack S or not.

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 and HIR pages is chosen such that all LIR pages and only a small percentage of HIR pages are kept in cache. 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. Now, if the new IRR of the requested page is smaller than the recency of some LIR page, then their LIR / HIR statuses are interchanged. Usually only around 1% of the cache is used to store HIR pages while 99% of the cache is reserved for LIR pages.

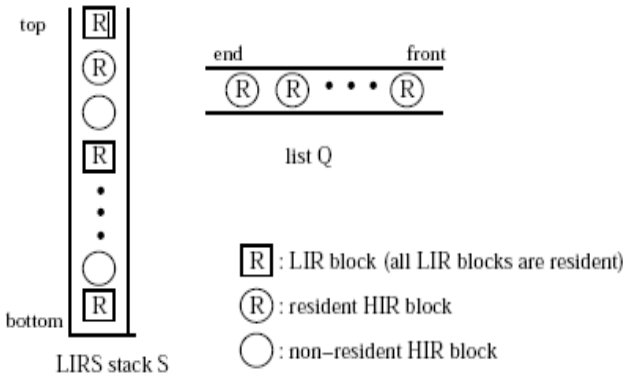


Figure 2.2 : The LIRS stack S holds LIR blocks as well as HIRS blocks with or without resident status, and a list Q holds all the resident HIR blocks.

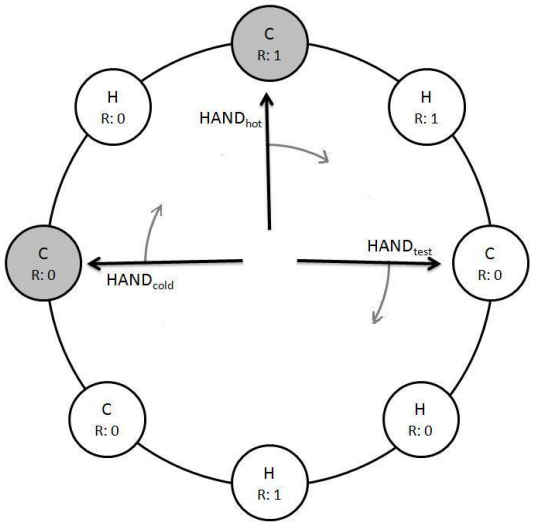
H. CLOCK-Pro

The CLOCK-Pro algorithm tries to approximate LIRS using CLOCK. Reuse distance, which is analogous to IRR of LIRS, is an important parameter for page replacement decision in CLOCK-Pro [4]. When a page is accessed, the reuse distance is the period of time in terms of the number of other distinct pages accessed since its last access. A page is categorized as a cold page if it has a large reuse distance or as a hot page if it has a small reuse distance.

Let the size of main memory be *m*. It is divided into hot pages (size: *mh*) and cold pages (size: *mc*). Apart from these, at most *m* non-resident pages have their history access information cached. Hence a total of *2m* meta-data entries are present for keeping track of page access history. A single list is maintained to place all the accessed pages (either hot or cold) in the order of page access. Naturally, the pages with small recency are at the list head and the pages with large recency are at the list tail.

After a cold page is accepted into the list, a test period is granted to that page. This is done to give the cold pages a chance to compete with the hot pages. If the page is re-accessed during the test period, it is turned into a hot page. On the other hand, if the page is not re-accessed, then it is removed from the list. A cold page in its test period can be removed out of memory; however, its page meta-data is kept in the list for the test purpose until the end of its test period.

The test period is set as the largest recency of the hot



pages. The page entries are maintained in a circular list. For each page, there are three status bits; a cold/hot indicator, a reference bit and for each cold page an indicator to determine if the page is in test period.

Fig. 2.3. Working of CLOCK-Pro

In CLOCK-Pro(see Fig. 2.3.), there are three hands. $HAND_{cold}$ points to the last resident cold page i.e., the cold page to be next replaced. During page replacement, if the reference bit of the page pointed by $HAND_{cold}$ is 0, the page is evicted. If the page is in the test period, its meta-data will be saved in the list. If the reference bit is 1 and the page is in test period, it is turned as a hot page and the reference bit is reset. $HAND_{cold}$ is analogous to the hand in the clock algorithm.

$HAND_{hot}$ points to the hot page with the largest recency. If reference bit of the page pointed by $HAND_{hot}$ is 0, the page is turned to cold page. If the reference bit is 1, then it is reset and the hand moves clockwise by one page. If the page is a non-resident cold page and its test period is terminated, the page is removed from the list. At the end, $HAND_{hot}$ stops at a hot page. At any point, if the number of non-resident cold pages exceeds *m*, the test period of the cold page pointed by $HAND_{test}$ is terminated and the page is removed from the list. This means that the cold page was not re-accessed during its test period and hence should be removed. HANDtest stops at the next cold page.

When a page fault occurs, the page is checked in the memory. If the faulted page is in the list as a cold page, it is turned into a hot page and is placed at the head of the list of hot pages. Also $HAND_{hot}$ is run to change a hot page with largest recency to a cold page. This is done to balance the number of hot and cold pages in the memory. If the faulted page is not in the list, it is brought in the memory and set as a cold page. This page is placed at the head of the list of cold pages and its test period is started. At any point if the number of cold pages exceed ($mc + m$), HANDtest is run to evict non-referenced cold pages.

Like LIRS, CLOCK-Pro is adaptive to access patterns with strong and weak locality. Previous simulation studies have indicated that LIRS and CLOCK-Pro provide better performance than LRU for a variety of memory access patterns.

I. Adaptive Replacement Cache (ARC)

The Adaptive Replacement Cache (ARC) is an adaptive page replacement algorithm developed at the IBM Almaden Research Center [5]. The algorithm keeps a track of both frequently used and recently used pages, along with some history data regarding eviction for both.

ARC maintains two LRU lists: L1 and L2. The list L1 contains all the pages that have been accessed exactly once recently, while the list L2 contains the pages that have been accessed at least twice recently. Thus L1 can be thought of as capturing short-term utility (recency) and L2 can be thought of as capturing long term utility (frequency). Each of these lists is split into top cache entries and bottom ghost entries. That is, L1 is split into T1 and B1, and L2 is split into T2 and B2. The entries in T1 union T2 constitute the cache, while B1 and B2 are ghost lists. These ghost lists keep a track of recently evicted cache entries and help in adapting the behavior of the algorithm. In addition, the ghost lists contain only the meta-data and not the actual pages.
The cache directory is thus organized into four LRU lists:

1. T1, for recent cache entries
2. T2, for frequent entries, referenced at least twice
3. B1, ghost entries recently evicted from the T1 cache,
4. B2, similar ghost entries, but evicted from T2

If the cache size is *c*, then $|T1 + T2| = c$. Suppose $|T1| = p$, then $|T2| = c - p$. The ARC algorithm continually adapts the value of parameter *p* depending on whether the current workload favors recency or frequency. If recency is more prominent in the current workload, *p* increases; while if frequency is more prominent, *p* decreases ($c - p$ increases).

Also, the size of the cache directory, $|L1| + |L2| = 2c$.

For a fixed *p*, the algorithm for replacement would be as:

1. If $|T1| > p$, replace the LRU page in T1
2. If $|T1| < p$, replace the LRU page in T2
3. If $|T1| = p$ and the missed page is in B1, replace the LRU page in T2
4. If $|T1| = p$ and the missed page is in B2, replace the LRU page in T1

The adaptation of the value of *p* is based on the following idea: If there is a hit in B1 then the data stored from the point of view of recency has been useful and more space should be allotted to store the least recently used one time data. Thus, we should increase the size of T1 for which the value of *p* should increase.

If there is a hit in B2 then the data stored from the point of view of frequency was more relevant and more space should be allotted to T2. Thus, the value of *p* should decrease. The amount by which *p* should deviate is given by the relative sizes of B1 and B2.[5]

J. CLOCK with Adaptive Replacement (CAR)

CAR attempts to merge the adaptive policy of ARC with the implementation 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. The concept behind these lists is same as that for ARC. In addition, the lists T1 and T2 i.e. the pages in the cache, have a reference bit that can be set or reset. Intuitively $T1^0$, B1indicate "recency"(Fig. 2.(a)) and $T1^1 U\ T2\ U\ B2$ indicate "frequency"(Fig. 2.(b)).

The precise definition of four lists is as follows:

1.$T1^0$ and $B1$ contains all the pages that are referenced exactly once since its most recent eviction from $T1$ U $T2$ U $B1$ U $B2$ or was never referenced before since its inception.

2. $T1^1$, $B2$ and $T2$ contains all the pages that are referenced more than once since its most recent eviction from $T1$ U $T2$ U $B1$ U $B2$.

The two important constraints on the sizes of T1, T2, B1 and B2 are:

1. $0 \leq |T1|+|B1| \leq c$.

By definition,$T1$ U $B1$ captures recency. The size of recently accessed pages and frequently accessed pages keep on changing. This prevents pages which are accessed only once from taking up the entire cache directory of size $2c$ since increasing size of $T1$ U $B1$ indicates that the recently referenced pages are not being referenced again which in turn means the recency data that is stored is not helpful. Thus it means that only the frequently used pages are re-referenced or new pages are being referenced.

2. $0 \leq |T2|+|B2| \leq 2c$.

If only a set of pages are being accessed frequently, there are no new references. The cache directory has information regarding only frequency.

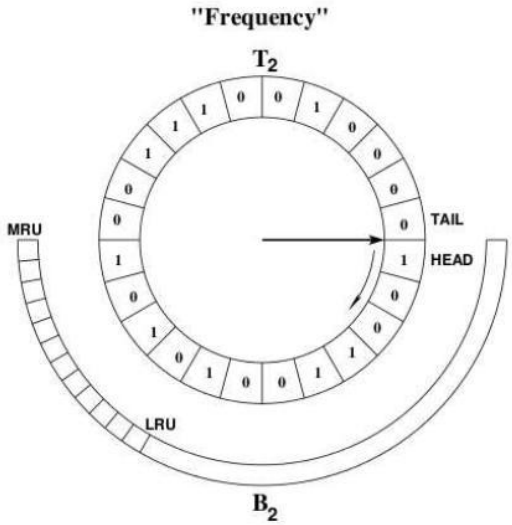


Fig : (a)

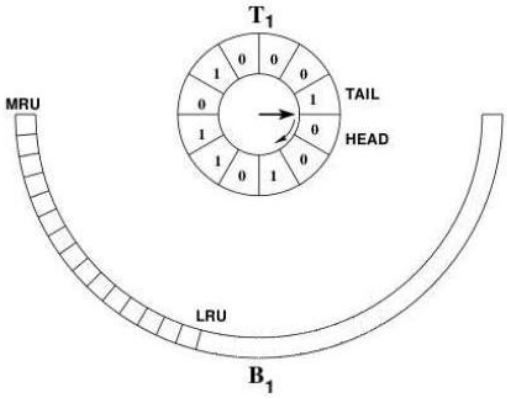


Fig : (b)

Fig. 2.4 : Working of Clock with Adaptive Replacement

Figure. 2.4 shows a visual description of CAR. The CLOCKS T1 and T2 contain those pages that are in the cache and the lists B1 and B2 contain history pages that were recently evicted from the cache. The CLOCK T1 captures "recency" while the CLOCK T2 captures "frequency." The lists B1 and B2 are simple LRU lists. Pages evicted from T1 are placed on B1, and those evicted from T2 are placed on B2. The algorithm strives to keep B1 to roughly the same size as T2 and B2 to roughly the same size as T1. The algorithm also limits |T1| + |B1| from exceeding the cache size. The sizes of the CLOCKs T1 and T2 are adapted continuously in response to a varying workload. Whenever a hit in B1 is observed, the target size of T1 is incremented; similarly, whenever a hit in B2 is observed, the target size of T1 is decremented. The new pages are inserted in either T1 or T2 immediately behind the clock hands which are shown to rotate clockwise. The page reference bit of new pages is set to 0. Upon a cache hit to any page in T1 ∪ T2, the page reference bit associated with the page is simply set to 1. Whenever the T1clock hand encounters a page with a page reference bit of 1, the clock hand moves the page behind the T2 clock hand and resets the page reference bit to 0. Whenever the T1 clock hand encounters a page with a page reference bit of 0, the page is evicted and is placed at the MRU position in B1. Whenever the T2 clock hand encounters a page with a page reference bit of 1, the page reference bit is reset to 0. Whenever the T2 clock hand encounters a page with a page reference bit of 0, the page is evicted and is placed at the MRU position in B2.[1]

The idea behind the algorithm is as follows. In case of a cache hit, the requested page is delivered from the cache and its reference bit is set. Instead, if the page is not in the cache but is present in list B1, this indicates the page had been used once recently before being evicted from the cache. This page is then moved to the head of T2 and its reference bit is set to 0. Also, a hit in B1 indicates that pages used once recently are required again implying a recency favoring workload.

Hence the value of p has to be increased resulting increase in the size of T1. Accordingly, if the page is not in the cache but in B2, this indicates that the page had been used frequently before being evicted from the cache. This page is then moved to the head of T2 and its reference bit is set to 0. Also, a hit in B2 indicates that the pages used frequently are required again implying a frequency favoring workload. Hence the value of p has to be decreased resulting increase in the size of T2.

Finally if the page is not found in B1 U B2, then the page is added to the MRU position in T1. In any of the above cases if the cache is full (|T1| + |T2| = c), then the CLOCK policy is applied on either T1 or T2 depending on the parameter p.

CONCLUSION

Modern computers often have some form of virtual memory. In the simplest form, each process' address space is divided up into uniform sized blocks called pages, which can be placed into any available page frame in memory. There are many page replacement algorithms: aging and WSClock. LRU-based working set size estimation is an effective technique to support memory resource management.

Paging systems can be modeled by abstracting the page reference string from the program and using the same reference string with different algorithms. These models can be used to make some predictions about paging behavior. To make paging systems work well, choosing an algorithm is not enough; attention to issues such as determining the working set, memory allocation policy, and page size are required.

This paper outlines that evolution of page replacement algorithms lead to efficient use of memory. These technique one by one are more applicable by significantly reducing its overhead. Experimental evaluation shows that, these algorithms are capable of reducing the overhead with sufficient precision to improve memory allocation decisions.

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] S. Jiang, and X. Zhang, "LIRS: An Efficient Policy to improve Buffer Cache Performance", *IEEE Transcations on Computers*, pp. 939-952, 2005.

[4] S. Jiang, X. Zhang, and F. Chen, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement", *ATEC '05 Proceedings of* the annual conference on USENIX Anuual Tecchnical Conference, pp. 35, 2005.

[5] N. Meigiddo, and D. S. Modha, "ARC: A Self-Tuning, Low overhead Replacement Cache", *IEEE Transactions on Computers*, pp. 58-65, 2004.

[6] 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.

[7] 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.

[8] Tanenbaum, Andrew Stuart., "Modern Operating Systems", Prentice Hall, 2001.

[9] Operating System Concepts 6th ed - Silberschatz Galvin

[10] "The Page Replacement with Working Set for Linux" by V. M. Aswar at National Conference on "Advances in Engg. And Management presentation, Indira College of Engineering and Management, Post: Parandwadi, Tal: Maval Pune. Jan. 2011.

AUTHOR'S PROFILE

Varsha M. Aswar is currently pursuing her Master.Degree In Computer Science and Engineering at the Prof. Ram Meghe Institute of Technology and Research, Amravati (India). Her areas of interest include Operating System, Computer Graphics.

Prof. Amol. P. Bodkhe is Professor in the Department Of Electronics and Telecommunication Engineering at Prof. Ram Meghe Institute of Technology & Research, Badnera – Amravati. He has done his M.E in Electronics and Telecommunication Engineering. He has started his work of Ph.D. He is having 27 years of teaching experience. His area of interest is communication.