A Comprehensive Survey of Page Replacement Algorithms

Mohd Zeeshan Farooqui, Mohd Shoaib, Mohammad Zunnun Khan

***Abstract*— Efficient page replacement algorithms are needed in virtual memory system to make a decision which pages to evict from memory in case of a page fault. Many algorithms have been proposed over the years. Each algorithm tries to incur minimum overhead while attempting to minimize the page fault rate. Formulating newer approaches of page replacement which could adapt to changing workloads have been the main focus of research as newer memory access patterns were explored. 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, Working Set, WSClock, Dynamic Sort.**

***Index Terms*— Page Replacement, LRU, Aging, Second Chance, Clock-Pro.**

I. Introduction

In order to completely understand the full potential of multi-programming systems it is essential to interleave the execution of more programs than they can be physically accommodated in main memory. Hence we use a two–level memory hierarchy consisting of a slower but cheaper secondary memory and a faster but costlier main 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 [1]. Pages are brought into main memory only when the executing process demands them and 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 fetch from secondary memory. In such a case an existing page in main memory needs to be discarded. The selection of such a page is performed by page replacement algorithms which try to minimize the page fault rate by incurring least overhead.
This paper outlines the major advanced page replacement algorithms along with their relative pros and cons. We start with basic algorithms such as Optimal, LRU, FIFO, Second Chance and CLOCK and move on to the more advanced Dueling CLOCK, LRU-K, LIRS, CLOCK-Pro, WSClock, Dynamic sort and LR+5LF algorithms.

Manuscript Received January 2014

Mohd Zeeshan Farooqui*, Computer Science & Engineering, Shri Ramswaroop Memorial University, Barabanki, India.*

Mohd Shoaib*, Computer Science & Engineering, Shri Ramswaroop Memorial University, Barabanki, India.*

Mohammad Zunnun Khan*, Assistant Professor, Computer Science & Engineering, Integral University, Lucknow, India.*

II. Page Replacement Algorithms

A. Optimal Algorithm

The optimal algorithm replaces the page in main memory which will not be accessed for the longest time. To do this, it needs to look into the future, thus it cannot be realized in reality. However, it is quite useful for comparison purposes in simulations, because in a simulated environment, it is possible to predefine the order in which pages are accessed. Using the optimal algorithm, you can see how real algorithms perform compared to the optimal.

B. First In First Out (FIFO)

The FIFO (first-in-first-out) algorithm is extremely simple: It removes the page from main memory with the earliest creation time. This can be implemented using a list: New pages are inserted at the head of the list, and the page at the tail is swapped out. Another implementation is using a ring (usually referred to as clock): Every time a page has to be replaced, the page the pointer points at is swapped out and at the same place the new page is swapped in. After this, the pointer moves to the next page. The FIFO algorithm's performance is rather bad.

C. Second Chance Algorithm

The second-chance algorithm is very similar to FIFO. However, it interferes with the accessing process. Every page has, in addition to its dirty bit, a referenced bit (r-bit). Every time a page is accessed, the r-bit is set. The replacement process works like FIFO, except that when a page's r-bit is set, instead of replacing it, the r-bit is unset, the page is moved to the list's tail (or the pointer moves to the next page, resp.) and the next page is examined. Second Chances performs better than FIFO, but it is still far from optimal. The operation of this algorithm, called second chance, is shown in Fig. 1.In Fig. 1(a) we see pages *A* through *H* kept on a linked list and sorted by the time they arrived in memory [10].

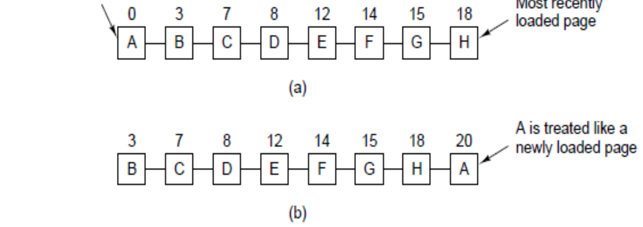


Figure 1- Operation of second chance. (a) Pages sorted in FIFO order. (b)Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their loading times.

Suppose that a page fault occurs at time 20. The oldest page is *A*, which arrived at time 0, when the process started. If *A* has the *R* bit cleared, it is evicted from memory, either by being written to the disk (if it is dirty), or just abandoned (if it is clean). On the other hand, if the *R* bit is set, *A* is put onto the end of the list and its ''load time'' is reset to the current time (20). The *R* bit is also cleared. The search for a suitable page continues with *B*.
What second chance is doing is looking for an old page that has not been referenced in the previous clock interval. If all the pages have been referenced, second chance degenerates into pure FIFO. Specifically, imagine that all the pages in Fig. 1(a) have their *R* bits set. One by one, the operating system moves the pages to the end of the list, clearing the *R* bit each time it appends a page to the end of the list. Eventually, it comes back to page *A*, which now has its *R* bit cleared. At this point *A* is evicted. Thus the algorithm always terminates.

D. CLOCK

Second chance is a reasonable algorithm, it is unnecessarily inefficient because it is constantly moving pages around on its list [2]. A better approach is to keep all the page frames on a circular list in the form of a clock, as shown in Fig. 2. A hand points to the oldest page.
When a page fault occurs, the page being pointed to by the hand is inspected. If its *R* bit is 0, the page is evicted [10], the new page is inserted into the clock in its place, and the hand is advanced one position. If *R* is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with *R* = 0. Not surprisingly, this algorithm is called **clock**. It differs from second chance only in the implementation.

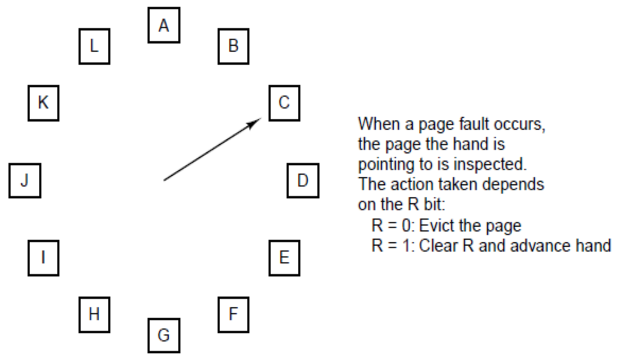

A
B
C
D
E
F
G
H
I
J
K
L
When a page fault occurs, the page the hand is pointing to is inspected. The action depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

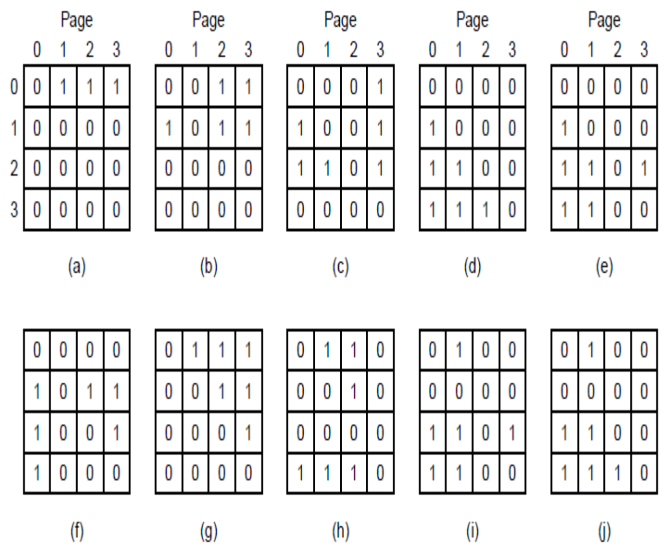
Figure 2- The clock page replacement algorithm

E. Not Recently Used (NRU)

The NRU (not-recently-used) algorithm uses an r-bit for every page. Every time a page is read, the r-bit is set. Periodically, all r-bits are unset. When a page fault occurs, an arbitrary page with r-bit unset is swapped out. NRU is actually Aging with a bit field width of 1, and it does not perform very well.

F. Least Recently Used (LRU)

The LRU policy is based on the principle of locality which states that program and data references within a process tend to cluster. The Least Recently Used replacement policy selects that page for replacement which has not been referenced for the longest time. 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.
Now let us look at a second hardware LRU algorithm. For a machine with *n* page frames, the LRU hardware can maintain a matrix of $n * n$ bits, initially all zero. Whenever page frame *k* is referenced, the hardware first sets all the bits of row *k* to 1, then sets all the bits of column *k* to 0. At any instant, the row whose binary value is lowest is the least recently used, the row whose value is next lowest is next least recently used, and so forth. The workings of this algorithm are given in Fig. 3 [10] for four page frames and page references in the order 0 1 2 3 2 1 0 3 2 3
After page 0 is referenced, we have the situation of Fig. 3(a). After page 1 is reference, we have the situation of Fig. 3(b), and so forth.



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0&1&1&1 \\ 1 & 0&0&0&0 \\ 2 & 0&0&0&0 \\ 3 & 0&0&0&0 \end{array}\quad
\begin{array}{cccc} 0&0&1&1 \\ 1&0&1&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{array}\quad
\begin{array}{cccc} 0&0&0&1 \\ 1&0&0&1 \\ 1&1&0&1 \\ 0&0&0&0 \end{array}\quad
\begin{array}{cccc} 0&0&0&0 \\ 1&0&0&0 \\ 1&1&0&0 \\ 1&1&1&0 \end{array}\quad
\begin{array}{cccc} 0&0&0&0 \\ 1&0&0&0 \\ 1&1&0&1 \\ 1&1&0&0 \end{array}$$

$$\begin{array}{cccc} 0&0&0&0 \\ 1&0&1&1 \\ 1&0&0&1 \\ 1&0&0&0 \end{array}\quad
\begin{array}{cccc} 0&1&1&1 \\ 0&0&1&1 \\ 0&0&0&1 \\ 0&0&0&0 \end{array}\quad
\begin{array}{cccc} 0&1&1&0 \\ 0&0&1&0 \\ 0&0&0&0 \\ 1&1&1&0 \end{array}\quad
\begin{array}{cccc} 0&1&0&0 \\ 0&0&0&0 \\ 1&1&0&1 \\ 1&1&0&0 \end{array}\quad
\begin{array}{cccc} 0&1&0&0 \\ 0&0&0&0 \\ 1&1&0&0 \\ 1&1&1&0 \end{array}$$

Figure 3- LRU using a matrix when pages are referenced in the order 0,1,2,3,2,1,0,3,2,3.

G. Aging

The aging algorithm is somewhat tricky: It uses a bit field of w bits for each page in order to track its accessing profile. Every time a page is read, the first (i.e. most significant) bit of the page's bit field is set. Every n instructions all pages' bit fields are right-shifted by one bit.

The next page to replace is the one with the lowest (numerical) value of its bit field. If there are several pages having the same value, an arbitrary page is chosen. The aging algorithm works very well in many cases, and sometimes even better than LRU, because it looks behind the last access. It furthermore is rather easy to implement, because there are no expensive actions to perform when reading a page. However, finding the page with the lowest bit field value usually takes some time. Thus, it might be necessary to predetermine the next page to be swapped out in background.

H. Dueling CLOCK

Dueling CLOCK is an adaptive cache replacement policy (ACR) that is based on the CLOCK algorithm. A major disadvantage of the regular CLOCK algorithm is that it is not scan resistant i.e. the page replacement algorithm that does not allow scanning to push frequently used pages out of main memory is said to be scan resistant [3]. To remove 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 prevailing algorithm for page replacement.

The only change required in order to make clock scan resistant is that the hand of the clock should 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 combination of two algorithms- 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 G2 always 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. A 10 bit policy select counter (PSEL) is used to determine the replacement policy for G3. Whenever a cache miss occurs on the cache set in group G1, the PSEL counter is decremented and whenever a cache miss occurs on the cache set in group G2, the counter is incremented. Now, group G3 adopts CLOCK replacement policy when the MSB of PSEL is 1, otherwise G3 uses the scan resistant CLOCK policy.

Dueling CLOCK algorithm has been shown to provide considerable performance improvement over LRU when applied to the problem of L2 cache management. [3]

I. LRU-K

The LRU policy takes into account the recency information while evicting pages, without considering the frequency. LRU-K was proposed to consider the frequency information, 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 Kth most recent reference to the page p. The policy considers Kth most recent reference to a page, it favors pages which are accessed frequently within a short time as pages referenced the smallest number of times would always be chosen first for page replacement.

Experimental results indicate that LRU-2 performs better than LRU [4]; while higher K does not result in an appreciable increase in the performance, but has high implementation overhead.

J. Low Inter-reference Recency Set (LIRS)

The Low Inter-reference Recency Set algorithm takes into consideration the Inter-Reference Recency of pages as the governing factor for eviction [5]. 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 Inter-Reference Recency (IRR) of a page is large, then the next IRR of the block is likely to be large again and hence the page is appropriate for removal as per Belady's MIN. It needs to be noted that the page with high IRR selected for removal may also have been recently used. The algorithm distinguishes between pages with high IRR (HIR) and 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 99% of the cache is reserved for LIR pages while 1% of the cache is used to store HIR pages.

K. CLOCK-Pro

ClockPro attempts to improve upon both LRU and Clock by considering frequency of access in addition to recency of access. To do so, it categorizes a block as either hot or cold. Hot blocks are accessed frequently, and cold blocks are not. The categorization is done live, and blocks can switch between the two categories. Clock-Pro also uses the same basic clock-bit and clock$_{hand}$ structure of the Clock algorithm. It also uses a test period. The test period tracks how old the block is in the cache. [6]

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: m_h) and cold pages (size: m_c). 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.

Now, 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 but 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. In CLOCK-Pro (see Fig. 4.), 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 removed. 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. $HAND_{test}$ 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 (m_c + m), $HAND_{test}$ is run to evict non-referenced cold pages.

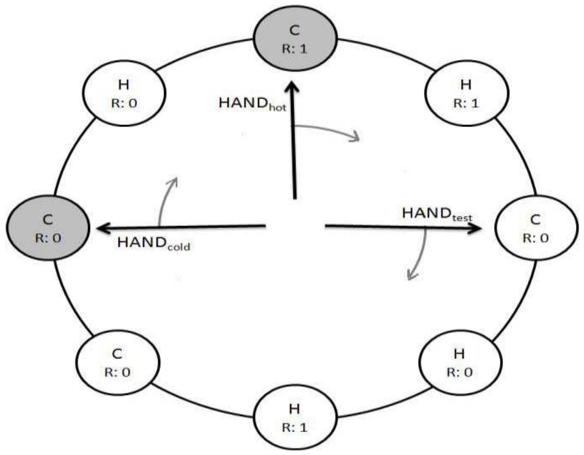


Figure 4- Working of CLOCK-Pro

L. The Working Set Page Replacement Algorithm

The working set of a process is the set of pages expected to be used by that process during some time interval. While the "working set model" isn't a page replacement algorithm in the strict sense (it's actually a kind of mid-term scheduler), several page replacement algorithms attempt to approximate the working set model, attempting to keep all pages in the working set in RAM.

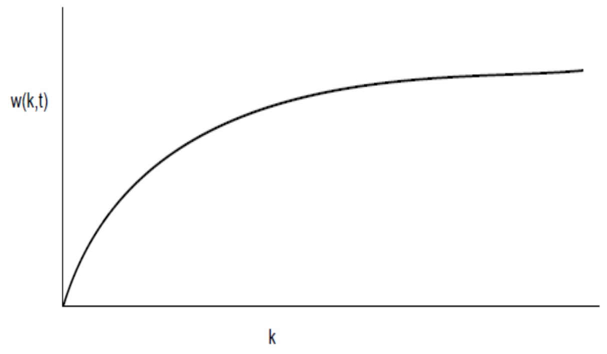


Figure 5- The working set is the set of pages used by the k most recent memory references. The function w(k, t) is the size of the working set at time t.

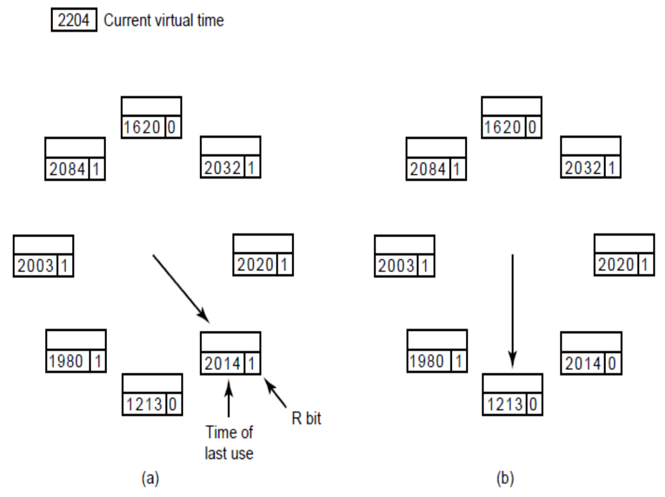
It has been long known that most programs do not reference their address space uniformly, but that the references tend to cluster on a small number of pages.

A memory reference may fetch an instruction, it may fetch data, or it may store data [10]. At any instant of time, *t*, there exists a set consisting of all the pages used by the *k* most recent memory references. This set, $w(k, t)$ is the working set. Because the $k = 1$ most recent references must have used all the pages used by the $k > 1$ most recent references, and possibly others, $w(k, t)$ is a monotonically non decreasing function of *k*. The limit of $w(k, t)$ as *k* becomes large is finite because a program cannot reference more pages than its address space contains, and few programs will use every single page. Figure .5 depicts the size of the working set as a function of *k*.

M. The WSClock Page Replacement Algorithm

The basic working set algorithm is burdensome since the entire page table has to be scanned at each page fault until a suitable candidate is located. An improved algorithm based on the clock algorithm but also uses the working set information is called WSClock**.** Due to its simplicity of implementation and good performance, it is widely used in practice. [7]

The data structure needed is a circular list of page frames, as in the clock algorithm, and as shown in Fig. 6(a). Initially, this list is empty. When the first page is loaded, it is added to the list. As more pages are added, they go into the list to form a ring. Each entry contains the *Time of last use* field from the basic working set algorithm, as well as the *R* bit (shown) and the *M* bit (not shown). As with the clock algorithm, at each page fault the page pointed to by the hand is examined first. If the *R* bit is set to 1, the page has been used during the current tick so it is not an ideal candidate to remove. The *R* bit is then set to 0, the hand advanced to the next page, and the algorithm repeated for that page. The state after this sequence of events is shown in Fig. 6(b). Now consider what happens if the page pointed to has *R* = 0, as shown in Fig. 6(c). If the age is greater than T and the page is clean, it is not in the working set and a valid copy exists on the disk. The page frame is simply claimed and the new page put there, as shown in Fig. 6(d).



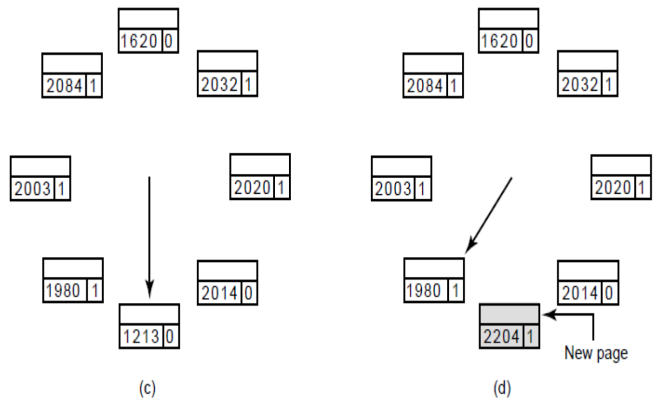


Figure 6- Operation of the WSClock algorithm. (a) and (b) give an example of what happens when R = 1. (c) and (d) give an example of R = 0.

On the other hand, if the page is dirty, it cannot be claimed immediately since no valid copy is present on disk. To avoid a process switch, the write to disk is scheduled, but the hand is advanced and the algorithm continues with the next page. After all, there might be an old, clean page further down the line that can be used immediately. In principle, all pages might be scheduled for disk I/O on one cycle around the clock. To reduce disk traffic, a limit might be set, allowing a maximum of n pages to be written back. Once this limit has been reached, no new writes are scheduled. What happens if the hand comes all the way around to its starting point?

There are two cases to distinguish:
1. At least one write has been scheduled.
2. No writes have been scheduled.

In the former case, the hand just keeps moving, looking for a clean page. Since one or more writes have been scheduled, eventually some write will complete and its page will be marked as clean. The first clean page encountered is evicted. This page is not necessarily the first write scheduled because the disk driver may reorder writes in order to optimize disk performance. In the latter case, all pages are in the working set, otherwise at least one write would have been scheduled. Lacking additional information, the simplest thing to do is claim any clean page and use it. The location of a clean page could be kept track of during the sweep. If no clean pages exist, then the current page is chosen and written back to disk.

N. Dynamic Sort Page Replacement

The Dynamic Sort Page Replacement policy is work based on the reference count. [8] The pages are collected at a certain point or time quantum. After that the collected pages are sorted based on their reference count. This process is repeated at each time quantum. The speed of this process of sorting depends on the processer speed. The page replacement algorithm processes only sorted pages. The main difficulty in this algorithm is that for each time quantum the reference string has to be sorted. In this process the first time the number of page faults are more but after sorting of pages and with increase in page frames page faults decrease.

O. Least Recently Plus Five Least Frequently Replacement Policy

LR+5LF is a replacement policy which is a combination of two popular replacement policies LRU and LFU [9]. In this first LRU and LFU weighing is done to obtain a balance between two policies by adding a weighing value to each of them by developing the LRU and LFU weighing algorithm.
Secondly combining of LRU and LFU is done which depends on the Weighing Least Recently Least Frequently Used (WLRFU) values, which determined by the following equation:

$$WLRFU[i] = WLRU[i]*C_r + WLFU[i]*C_f \qquad (1)$$

where C_r and C_f are priority constants for LRU and LFU respectively.
Thirdly the line to be replaced is determined, in normal case the line with minimum WLRFU will be replaced, but in case, where two or more blocks have the same WLRFU value or four ways set mapping, determining the line to be replaced algorithm should be used to resolve this problem.

III. Summary Of Page Replacement Algorithms

We have now looked at a variety of page replacement algorithms. In this paper we will briefly summarize them. The list of algorithms discussed is given in table below.

Table 1- Summary of Replacement Algorithms

Algorithm	**Summary**
Optimal	Not implementable but useful as benchmark
FIFO (First In First Out)	Might throw out important pages
NRU(Not Recently Used)	Very raw and unimportant
Second Chance	Big improvement over FIFO
Clock	Realistic
LRU(Least Recently Used)	Excellent but difficult to implement exactly
Aging	Efficient algorithm that approximates LRU better
LIRS and Clock-pro	provide better performance than LRU for various memory access pattern
Working Set	Somewhat Expensive to implement
WSClock	Good Efficient algorithm
Dynamic Sort	Efficient than FIFO
LR+5LF	Efficient than FIFO and LRU

The optimal algorithm replaces the page which will not be accessed for the longest time. Unfortunately, there is no way to predict which page will be last, so in practice this algorithm cannot be used. It is useful as a benchmark against which other algorithms can be measured.
The NRU algorithm divides pages into four classes depending on the state of two status bits *R* and *M*. A random page from the lowest numbered class is chosen. This algorithm is easy to implement, but it is unrefined. Better algorithms do exist.
FIFO keeps track of pages in the order they are loaded into memory by keeping them in a linked list. Removing the oldest page then becomes a hard decision as the page might still be in used in future, so FIFO is a bad choice.
Second chance is a modification to FIFO that checks if a page is in use before removing it. If it is, the page is spared. This modification greatly improves the performance.
Clock implements the second change so that pages are kept in a circular list with a pointer to the oldest page. It has the same performance characteristics, but takes a little less time to execute the algorithm.
LRU is an excellent algorithm which throws out page that hasn't been used in longest time by looking at past to work out the future but it cannot be implemented without special hardware. If this hardware is not available, it cannot be implemented.
Like LIRS, CLOCK-Pro is adaptive to access patterns with strong and weak locality. Studies have indicated that LIRS and CLOCK-Pro provide better performance than LRU for a variety of memory access patterns.
The working set algorithm is reasonable performance, but it is somewhat expensive to implement. WSClock is a variant of Clock that not only gives good performance but is also efficient to implement.
Dynamic Sort works by sorting reference string at certain time quantum. Provides better results than FIFO and LRU as page frames are increased. Lastly LR+5LF combine both LRU and LFU in 3 steps to provide better results than FIFO and LRU.
All in all, the best algorithms are CLOCK-Pro and WSClock. They are based on LRU and the working set. All give good paging performance and can be implemented efficiently.

REFERENCES

[1] 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.

[2] Sami Khuri, Hsiu-Chin Hsu, " Visualizing the CPU Scheduler and Page Replacement Algorithms", ACM 1-581 13.085-6/99/0003, 1999.

[3]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.

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

[6] 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 Annual Technical Conference, pp. 35, 2005.

[7] Richard W. Cart , John L. Hennessy, "WSClock - A Simple and Effective Algorithm for Virtual Memory Management", ACM 0-89791-062-1-12/81-0087, 1981.

[8] Abdulla Shaik, "Dynamic Sort (DS) Page Replacement", The International Journal of Computer Science & Applications, Volume 1, No. 3, May 2012 ISSN – 2278-1080.

[9] Adwan AbdelFattah, Aiman Abu Samra, "Least Recently Plus Five Least Frequently Replacement Policy (LR+5LF)", The International Arab Journal of Information Technology, Vol. 9, No. 1, January 2012.

[10] Andrew S. Tanenbaum, Albert S. Woodhull, "Operating Systems Design and Implementation", Second Edition, pages 400-406.

Mohd Zeeshan Farooqui has 2 years of experience in teaching different subjects in the field of Computer Science & Engineering and is currently pursuing his M.Tech. degree in Computer Science & Engineering from Shri Ramswaroop Memorial University (S.R.M.U), U.P. (India) and has completed B. Tech. in Computer Science & Engineering from Integral University, U.P. (India).

Mohd Shoaib has 3 years of experience in teaching various subjects in the field of Computer Science & Engineering and is currently pursuing his M.Tech. degree in Computer Science & Engineering from Shri Ramswaroop Memorial University (S.R.M.U), U.P. (India) and has completed B. Tech. in Computer Science & Engineering from Integral University, U.P. (India).

Mohammad Zunnun Khan has over 4 years of experience in teaching in Computer Science and Engineering. Presently associated with Integral University, Lucknow as Assistant Professor (Computer Science and Engineering Dept.), and taught different subjects at Under Graduate level as well as Post Graduate level. And also associated with different Conferences and refereed journals as Reviewer and Editor.