Page Replacement Algorithms in Memory Management: a survey

Saeid Taghavi Afshord, Mehdi Ayar

Department of Computer Engineering, Shabestar Branch, Islamic Azad University, Shabestar, Iran

taghavi@iau.ac.ir (Corresponding Author); mehdi.ayar@gmail.com

Abstract: One of the most important resources in a computer system is memory. Processes cannot run unless their code and data structures are in RAM. Memory management is important and is the most complex task for an operating system. Page replacement policies have been under extensive study over the years. A large number of different page replacement algorithms have been proposed and many of them have been implemented in operating systems and database management systems. The page fault rate has critical criteria for choosing suitable page replacement algorithms. In this paper, we describe algorithms that are widely simulated and utilized in practice. Then, we indicate the effective and efficient algorithm among them.

Keywords: memory, operating system, page replacement, performance and page faults

1. Introduction

Virtual memory refers to the technology in which some space in the hard disk is used as an extension of main memory so that a program need not worry if its size exceeds the size of the main memory. If It does happen, only a part of the program will reside in the main memory and other parts will remain on the hard disk and may be switched into memory later.

This mechanism is similar to the two-level memory hierarchy discussed in [1], including cache and main memory because the principle of locality is also a basis here. With virtual memory, if a piece of the process needed is not in the main memory, another piece will be swapped out and the former be brought in. If the latter is used immediately, then it will load back into the main memory right away. As we know, access to a hard disk is time-consuming compared to access to the main memory, Thus the reference to the virtual memory space on hard disks will deteriorate the system performance significantly. Fortunately, the principle of locality holds. It is the instruction and data references during a short period that tend to be bound to one piece of the process. So, access to hard disks will not be frequently requested and performed. Thus, the same principle, on the one hand, enables the caching mechanism to increase system performance, and on the other hand, avoids the deterioration of performance with virtual memory.

With virtual memory, there must be some facility to separate a process into several pieces so that they may reside separately either on hard disks or in main memory. Paging and/or segmentation are two methods that are usually used to achieve the goal.

2. Replacement policies

The replacement policy deals with the selection of a memory page to be replaced after a page fault occurs and a new page must be brought in. The goal of any replacement algorithm includes two aspects: (1) The algorithm itself should be simple to implement and efficient to run; and (2) the selection of page should not harm the performance of the virtual system as a whole, or more specifically, the page that is removed should be the page least likely to be referenced shortly.

2.1. Algorithms

The most important page replacement algorithms have been considered in the literature explained in this survey.

2.1.1. Optimal

The Optimal algorithm says that we should always replace the page that will not be used for the longest period, which means we have to be able to predict the future. It is possible that we can predict the future for certain applications with extremely regular page access patterns. However, in general, the complexity of applications, the dynamic environment of machines especially multiprocessing systems, and other random factors such as user interactions make the future very unpredictable.

The value of discussing this algorithm is that it may be a benchmark to evaluate the performance of other algorithms.

2.1.2. Least recently used (LRU)

Although we do not know the future exactly, we can predict the future to some extent based on the history. Based on the principle of locality, the page that has not been used for the longest time is also least likely to be referenced. The LRU algorithm thus selects that page to be replaced.

And experience tells us that the LRU policy does nearly as well as the optimal policy. However, since the decision-making is based on history, the system has to keep the references that have been made from the beginning of the execution. The overhead would be tremendous.

2.1.3. First in first out (FIFO)

The FIFO policy treats the page frames allocated to a process as a circular buffer, and pages are removed in a round-robin style. It may be viewed as a modified version of the LRU policy, and this time instead of the least recently used, the earliest used page is replaced since the page that has resided in main memory for the longest time will also be least likely used in the future.

This logic may be wrong sometimes if some part of the program is constantly used, which thus may lead to more page faults. The advantage of this policy is that it is one of the simplest page replacement policies to implement since all that is needed is a pointer that circles through the page frames of the process.

The Table 1 shows the comparison of these page replacement algorithms on various parameters made in the survey. The parameters considered are principle, performance, page faults and memory usage.

2.1.4 Second Chance

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) and the next page is examined. Second Chances performs better than FIFO, but it is still far from optimal.

2.1.5 Aging

The aging algorithm is somewhat tricky: It uses a bit field of *w* bits for each page to track its accessing profile. Every time a page is read, the *first* (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 several pages are 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 the background.

Table1. Comparison of three page replacement algorithms

Parameter	*Methods*		
	FIFO	LRU	Optimal
Principle	Replaces the oldest page in memory	Replaces the least recently used page	Replaces the page not needed for the longest time in the future
Complexity	Simple implementation	Moderate complexity due to tracking recency	High complexity, requires future knowledge
Data Structures	Queue	Stack, linked list, or counters	Not applicable in real-world scenarios
Performance	Generally suboptimal, suffers from Belady's anomaly	Better than FIFO, does not suffer from Belady's anomaly	Best theoretical performance, lowest page faults
Memory Usage	Low	Moderate, additional space for tracking usage history	Low(theorical)
Page Faults	High	Moderate to low	Lowest possible
Predictability	Poor, unpredictable due to anomaly	Predictable, stable performance	Theoretically predictable
Temporal Locality	Not considered	Considers temporal locality	Assumes perfect knowledge of future references
Optimal Conditions	Suitable for simple, predictable patterns	Suitable for varied, realistic access patterns	Serves as a benchmark, not used in practical systems

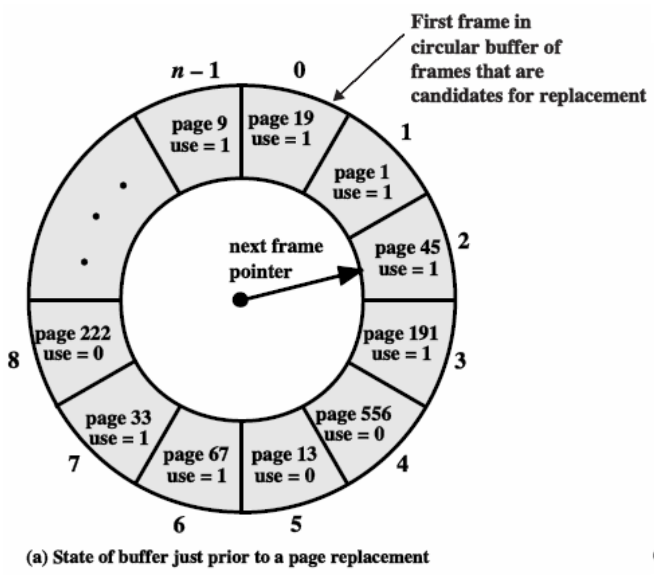
2.1.6 Clock

Each of the above policies has its advantages and disadvantages. Some may need less overhead, and some may produce better results. Thus here is an issue of balance. People have proposed all kinds of algorithms based on different considerations of balance between overhead and performance. Among them, the clock policy is one of the most popular ones.

The clock policy is a variant of the FIFO policy, except that it also considers to some extent the last accessed times of pages by associating an additional bit with each frame, referred to as the use bit. And when a page is referenced, its use bit is set to 1.

As Figure 1 illustrates, the set of frames that might be selected for replacement is viewed as a circular buffer, with which a pointer is associated. When a free frame is needed but not available, the system scans the buffer to find a frame with a use bit of 0 and the first frame of this kind will be selected for replacement. During the scan, whenever a frame with a use bit of 1 is met, the bit is reset to 0. Thus if all the frames have a use bit of 1, then the pointer will make a complete cycle through the buffer, setting all the use bits to 0, and stop at its original position, replacing the page in that frame. After a replacement is made, the pointer is set to point to the next frame in the buffer.

Figure 1 gives an example of this clock policy. Figure 1 shows the status of the buffer at some moment. Suppose page 727 is referenced but is not available in the buffer which is already full. Thus a page fault occurs and a page needs to be replaced. According to the clock policy, page 556 is replaced. The other updates include the use of bits of frame and 3 are set to 0 and the pointer is made pointing to frame 5.



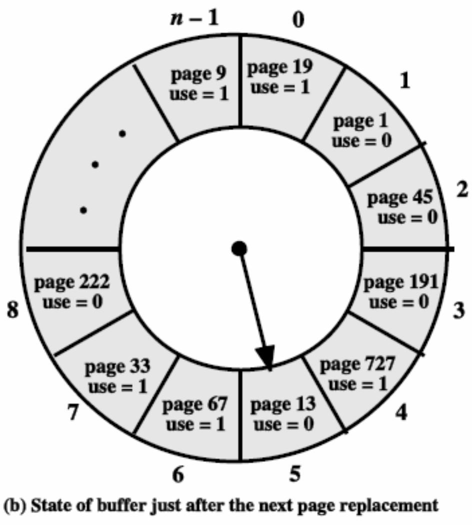


Fig.1. Example of clock policy operation

2.1.7. 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.

2.1.8 Not Frequently Used (NFU)

Is a software approximation to LRU: a perframe counter is incremented every clock tick if the frame has been referenced since the last clock tick; the frame with the lowest count is evicted on page fault. NFU does not accurately reflect temporal locality; a frame frequently accessed a long time ago will be kept while a frame accessed more recently but fewer times will be evicted.

2.1.9. LRU-K

The algorithm is called the LRU-K method and reduces to the well-known LRU (Least Recently Used) method for K=1. At first, it has shown the effectiveness of K > 1 by simulation [3], especially in the most common case of K = 2. The basic idea in LRU-K is to keep track of the times of the last K references to the memory pages and to use this statistical information to rank-order the pages as to their expected future behavior.

Based on this, the page replacement policy decision is made: which memory-resident page to replace when a newly accessed page must be read into memory. In [3] proved, under the assumptions of the independent reference model, that LRU-K is optimal among all replacement algorithms that can be based on information about K most recent references to each page. The proof uses the Bayesian formula to relate the space of actual page probabilities of the model to the space of observable page numbers on which the replacement decision is made.

In [3] had analyzed the LRU-K algorithm. The use of Bayesian methods in that analysis is, to our knowledge, a new and entirely appropriate way of handling the real lack of knowledge of page identity at the memory buffer level of the software.

2.2 MS, Application-specific algorithm

Different applications have different memory reference patterns. Most operating systems, however, are oblivious to this simple truth. Some algorithms are highly suitable for certain kinds of applications but inadequate for other kinds of workloads. Some of the operating system designers and theoreticians such as [4] believe there is a range of applications can benefit from application-specific memory management policies. Further, It observed that reference patterns do change during process execution, suggesting that applications can further benefit from dynamic selection of a suitable VM (Virtual Machine) policy [4].

For more information about this type of algorithm and as a sample, refer to [4] where they described the algorithm called MS, with full expressions of architecture, implementation, and test and performance evaluation of it.

3. Linux mechanism

Linux uses a hybrid of LFU and LRU algorithms, but it behaves approximately like LRU. Linux keeps a page age counter for each physical page in memory to inform the Kernel Page Swap Daemon (KSWAPD) whether a page is worth swapping out. The page age can be between 0 and 20 where 0 is the oldest and 20 is the youngest. When initially allocated, a page is given an age of 3, and each time the page is referenced, its age increases by 3 to a maximum of 20. KSWAPD round robins through each virtual page of each process. If a virtual page resides in physical memory and has not been referenced since the last KSWAPD scan, it will decrease its age by 1 to a minimum of 0. Pages with 0 age are candidates for swapping, and a further test on the dirty bit in its page table entry and its page priority will decide if the page should be swapped out. By using this page replacement mechanism, Linux favors, and, is likely to keep in memory the pages that have been accessed recently and frequently during a past period.

4. Conclusion and discussion

Unfortunately, there is no way to determine which page will be last, so, the optimal algorithm cannot be used practically. However, 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 the *R* and *M* bits. A random page from the lowest-numbered class is chosen. This algorithm is easy to implement, but it is very crude. Better ones exist. FIFO keeps track of the order of pages loaded into memory by keeping them in a linked list. Removing the oldest page then becomes trivial, but that page might still be in use, so FIFO is a bad choice. The 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. The clock is simply a different implementation of the second chance. It has the same performance properties but takes less time to execute the algorithm. LRU is an excellent algorithm, but it cannot be implemented without special hardware. If this hardware is not available, it cannot be said. NFU is a crude attempt to approximate LRU. It is not very good. However, aging is a much better approximation to LRU and can be implemented efficiently. It is a good choice.

Consider the reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. The page replacement algorithms of FIFO, LRU and Optimal will be applied using 3 frames and number of page faults will be calculated (M=miss and H= hit). With these assumptions, the number of page faults in FIFO, LRU, and Optimal algorithms using three frames is calculated as follows:

Number of page faults in FIFO using 3 frames: 15

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
M	M	M	M	H	M	M	M	M	M	M	H	H	M	M	H	H	M	M	M

Number of page faults in LRU using 3 frames: 12

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
M	M	M	M	H	M	H	M	M	M	M	H	H	M	H	M	H	M	H	H

Number of page faults in Optimal using 3 frames: 09

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
M	M	M	M	H	M	H	M	H	H	M	H	H	M	H	H	H	M	H	H

Consider the reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. The page replacement algorithms of FIFO, LRU and Optimal will be applied using four frames and number of page faults will be calculated. M=miss and H= hit.

Number of page faults in FIFO using 4 frames: 10

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7	7	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2
	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	7	7	7
		1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
			2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
M	M	M	M	H	M	H	M	H	H	M	H	H	M	M	H	H	M	H	H

Number of page faults in LRU using 4 frames: 08

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
M	M	M	M	H	M	H	M	H	H	H	H	H	M	H	H	H	M	H	H

Number of page faults in Optimal using 4 frames: 08

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7	7	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	7	7	7
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
M	M	M	M	H	M	H	M	H	H	H	H	H	M	H	H	H	M	H	H

Considering the above sequence of 20 pages reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 for memory frames of three and four pages, the page trace is given to the three replacement algorithms namely FIFO, LRU and optimal page replacement and the page faults are observed. The process is implemented on Windows 10 64-bit operating system using Java.

The results obtained for page faults using FIFO, LRU and Optimal algorithms using 3, 4 and 5 frames is shown in Table 2. The comparative analysis of FIFO, LRU, and Optimal page replacement algorithms demonstrates that while the optimal algorithm provides the best theoretical performance, LRU offers a practical and efficient alternative. FIFO, though easy to implement, generally performs worse due to its simplistic approach. Optimal emerges as the best-suited algorithm for real-world applications, balancing performance and practicality effectively.

Table 2. Page Faults Obtained using FIFO, LRU and Optimal Algorithms

Frame Size	*Methods*		
	FIFO	LRU	Optimal
3	15	12	09
4	10	08	08
5	09	07	07

Some other algorithms based on counting mechanisms that we did not introduce in this paper (see [2] for detailed information), such as Least Frequently Used (LFU) and Most Frequently Used (MFU) algorithms, have also been studied but not very widely used due to both their poor performance and large space requirement. The Most Recently Used (MRU) algorithm has been shown to perform well for a certain class of applications, such as database systems with large sequential access.

The Linux LRU algorithm performs well for many general applications. However, other algorithms have their specialty area of applications. One obvious example is MRU, which always swaps out the just referenced pages. In cases of sequential access or random access to pages that nowadays don't reference, MRU performs much better than the common LRU algorithm. LRU-K, which keeps track of the last K accesses in history instead of just the last access as in normal LRU for each page. It has been shown to be optimal under the assumption of the independent reference model, given the same amount of information about past page accesses.

Finally, the two best algorithms are aging and WSClock (This algorithm is described in [1] and we do not discuss it in this paper). They are based on LRU and the working set, respectively. Both give good paging performance and can be implemented efficiently. A few other algorithms exist, but these two are probably the most important in practice. We summarized the discussed algorithms in Table 3.

Table 3. Summary of the Page replacement algorithms discussed in the text

Algorithm	*Comment*
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important page
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
sNFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
LRU-K	The best, more difficult to implement of LRU

5. References

[1] A. S. Tanenbaum,, Modern Operating Systems, Prentice Hall; Second edition, 2001.

[2] W. Stallings. Operating Systems Internals and Design Principles, Fifth Edition, Prentice Hall, 2004.

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

[4] S. Chang, K. Zhang, Application Specific Memory Management, Electrical Engineering and Computer Science Department, University of California, Berkeley.

[5] S. H. Abbas, W. A. K. Naser, and L. M. Kadhim, "Study and Comparison of Replacement Algorithms," Int. J. Eng. Res. Adv. Technol., vol. 08, no. 08, pp. 01–06, 2022, doi: 10.31695/ijerat.2022.8.8.1.

[6] M. Waqar, A. Bilal, A. Malik, and I. Anwar, "Comparative analysis of replacement algorithms techniques regarding to technical aspects," Eur. J. Eng. Technol., vol. 4, no. 5, pp. 60–82, 2016.

[7] B. A. Tingare and V. L. Kolhe, "Analysis of Various Page Replacement Algorithms in Operating System," Int. J. Sci. Res., vol. 5, no. 12, pp. 578–584, 2016, [Online]. Available: https://www.ijsr.net/archive/v5i12/ART20163405.pdf.

[8] G. Rexha, E. Elmazi, and I. Tafa, "A Comparison of Three Page Replacement Algorithms: FIFO, LRU and Optimal," Acad. J. Interdiscip. Stud., vol. 4, no. 2, pp. 56–62, 2015, doi: 10.5901/ajis.2015.v4n2s2p56.

[9] H. M. H. Owda, M. A. Shah, A. I. Musa, and M. I. Tamimy, "A Comparison of Page Replacement Algorithms in Linux Memory Management," Int. J. Comput. Inf. Technol., vol. 03, no. 03, pp. 565–569, 2014, [Online]. Available: www.ijcit.com565.

[10] R. K. Gupta and M. A. Franklin, "Working Set and Page Fault Frequency Paging Algorithms: A Performance Comparison," IEEE Trans. Comput., vol. c–27, no. 8, pp. 706–712, 1978.

[11] A. J. Smith, "A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory," IEEE Trans. Softw. Eng., vol. SE-4, no. 2, pp. 121–130, 1978, doi: 10.1109/TSE.1978.231482.

[12] S. Iranit, A. R. Karlin, and S. Phillips, "Strongly Competitive Algorithms for Paging with Locality of Reference," Soc. Ind. Appl. Math., vol. 25, no. 3, pp. 477–497, 1996, doi: 10.1007/springerreference_65225.