Volume 4, Issue 7, July 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Analysis on Page Replacement Algorithms with Variable Number of Frames

Manisha Koranga*
Department of Computer Science & Engineering Banasthali University ,Jaipur,India

Nisha Koranga
Department of Electronics &communication Engineering rachna college of engineering Manav, Faridabad,India

Abstract— In a Computer operating system that uses paging for virtual memory management, page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. In this paper the motive is to reviewing and comparing the basic page replacement algorithms. The main purpose of this paper is to analyse the effect of increasing number of frames on the page fault and to determine hit ratio in each case. The algorithms are implemented in C++ language and the results have been simulated using MATLAB.

Keywords— Virtual Memory, Paging, Page Replacement, Hit Ratio, fault rate.

I. INTRODUCTION

In a computer operating system that uses paging for virtual memory management, **page replacement algorithms** decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Paging happens when a **page fault** occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the quality of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself. [8] The operating system divides virtual address space into units called pages. Main memory is also divided to fixed size units called page frames. Each used page can be either in secondary memory or in a page frame in main memory. Naturally neither of these memories is needed for the pages, in the virtual address spaces of processes that are not used. A CPU generated address is called logical address or virtual address, where as memory management unit generated address is known as physical address. Before using this logical address, it must be translated to its corresponding physical address. This address translation has been done corresponding to every memory reference, so it is important that it must be fast. A special hardware unit referred to as Memory Management Unit (MMU) is used for such translation. MMU uses address mapping information which is usually located in page tables, to make the translation. If the given virtual address is not mapped to main memory, operating system is trapped by the MMU. This trap is called as page fault which gives an opportunity to the operating system to bring the desired page from secondary memory to main memory, and then update the page table correspondingly. In simple words we can say that-When the processor need to execute a particular page and main memory does not contain that page, this situation is known as PAGE FAULT. As each and every process has its own virtual address space, the operating system must keep track of all pages and the location of each page used by each process. When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. The cache checks for the contents of the requested memory location in any cache lines that might contain that address. If the processor finds that the memory location is in the cache, a cache hit has occurred. However, if the processor does not find the memory location in the cache, a cache miss has occurred. In the case of:

1.a cache hit, the processor immediately reads or writes the data in the cache line
2.a cache miss, the cache allocates a new entry, and copies in data from main memory; then, the request is fulfilled from the contents of the cache.

Hit ratio = Total number of Hit Counts / Total number of Reference Counts
To represent it as a percentage:
Hit % = Hit ratio * 100

II. DEMAND PAGING

A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper called pager. When a process is to be brought in, then the pager guesses which pages will be used before the process is swapped

out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed. Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Marking a page will have no effect if the process never attempts to access the page. While the process executes and accesses pages that are memory resident, execution proceeds normally. Access to a page marked invalid causes a **page-fault trap**. [8] This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following

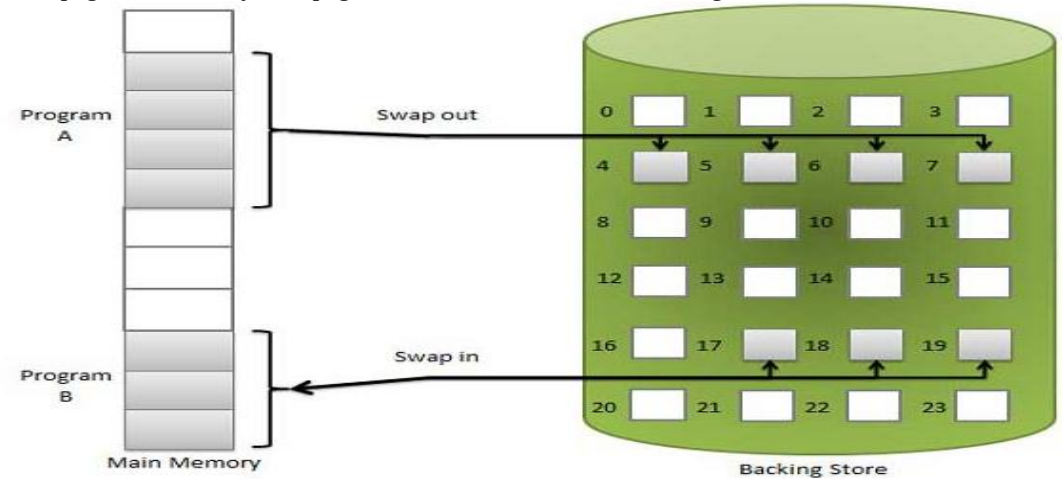


Fig1. Process executes and accesses pages that are memory resident

Access to a page marked invalid causes a **page-fault trap**. This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following

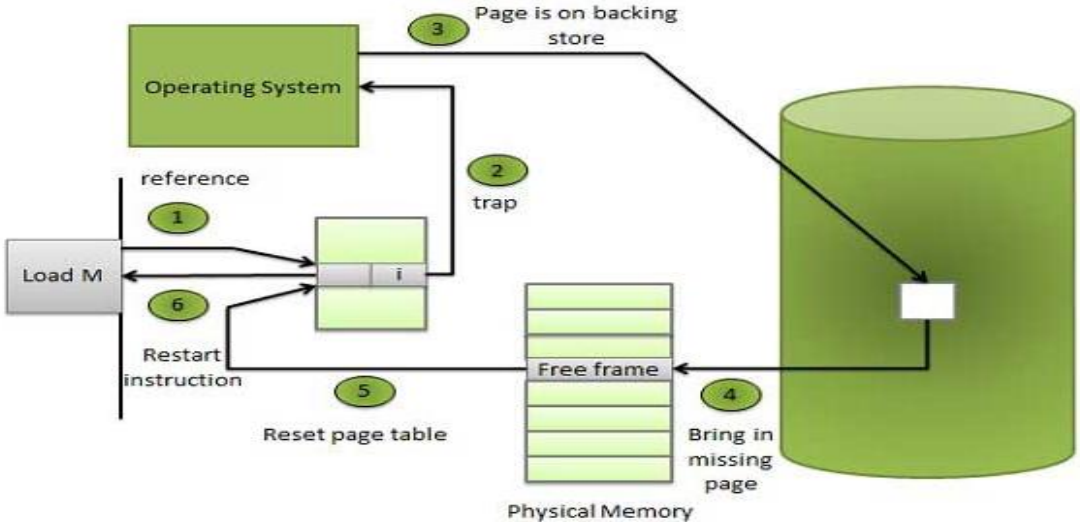


Fig.2 Steps in Handling a Page Fault

1. Check an internal table for this process, to determine whether the reference was a valid or it was an invalid memory access.
2. If the reference was invalid, terminate the process. If it was valid, but page have not yet brought in, page in the latter.
3. Find a free frame.
4. Schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory. Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

III. THE PAGE REPLACEMENT ALGORITHMS

Page replacement algorithms are the techniques using which Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages. When the page that was selected for replacement and was paged out, is referenced again

then it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm. A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults.

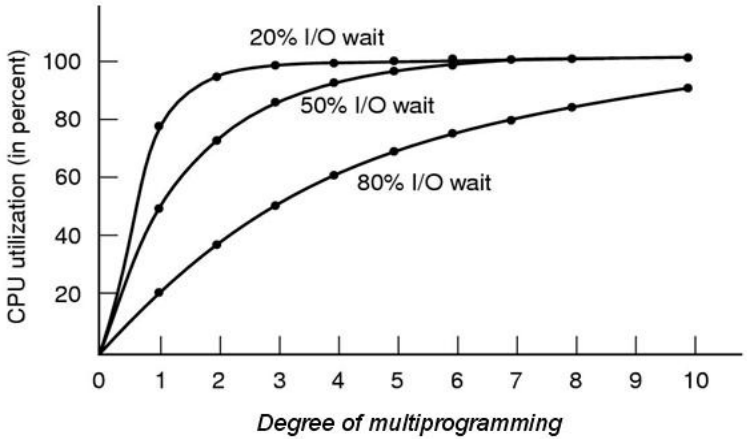


Fig. 3: Increasing CPU performance by I/O reducing

Performance of demand paging*:* Let—

a □ memory access time (usually 50–500 ns.),

p □ probability that a memory reference causes a page fault, and

s □ time to service a page fault.

Then the **effective access time** *(*EAT) = Hit Rate x Hit Time + Miss Rate x Miss Time

EAT* = (1–*p*) *a* + *ps

Major components of a page-fault service time are—

- Time to service the page-fault interrupt.
- Time to swap in the desired page.
- Time to restart the faulting process

There are a variety of page replacement algorithms. Some of them are described as follows:-

3.1 First in, first out (FIFO)

The first-in first-out algorithm is the simplest and oldest algorithm. The idea behind FIFO is to replace a page that is the oldest page in main memory from all the pages. FIFO focuses on the length of a time a page has been in memory rather than how much the page is being used". This requires implementation of a FIFO queue in which pages are ordered by arrival time. This is not always a good algorithm, because a page that is only used once remains in memory as long as a page that is used heavily. It also suffers from Be lady's anomaly: sometimes increasing the size of memory increases the fault rate. [6,8]

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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
✓	✓	✓	✓		✓	✓	✓	✓	✓	✓			✓	✓			✓	✓	✓

Fig 4: FIFO representation for 3 frames

3.2 The Optimal Algorithm

The Optimal page replacement (OPT) has the best performance, among all page replacement an algorithm i.e. has the lowest page fault rate of all the algorithms but its implementation is not realistic till now. The basic idea behind this algorithm is to replace the page that will never be needed again, or at least will be needed furthest in the future. Priority lists are used throughout. This algorithm simply replaces the page that will not be used for the longest period of time. So it is difficult to implement because it requires future knowledge of strings.

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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	0	4	4	4	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
✓	✓	✓	✓		✓		✓			✓			✓				✓		

Fig.5: optimal algorithm representation for 3 frames

However, one of reasons for designing such an algorithm, while it is not relevant practically, is its utilization as an ideal reference for assessing and comparing practical(and not theoretical) algorithms.

3.3 least-recently-used (LRU)

This algorithm [4] replaces the page that has not been used for the longest period of time. In general, LRU algorithm results better than FIFO algorithm. The reason behind this is that LRU takes into account the patterns of program behaviour by presuming that in most distant past used page is slightest likely to be used again later. The least-recently-used algorithm looks backward in the period of time. However, implementation of the LRU algorithm is now possible but it imposes large overhead to the system. [1]

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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
✓	✓	✓	✓		✓		✓	✓	✓	✓			✓		✓		✓		

Figure.6: LRU representation for 3 frames

IV. PROPOSED WORK

The Purpose of this paper is to determine the number of page faults, and hence the page-fault rate and corresponding hit ratio for a random reference string.

A page trace (or reference string) is a list of pages in order of their reference by the processor. Suppose the reference string is:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For a particular size memory, let's simulate which pages would be in memory for any replacement algorithm.

1.1 FIFO—Remove the page that has been in memory the longest. This requires implementation of a FIFO queue in which pages are ordered by arrival time. This is not always a good algorithm, because a page that is only used once remains in memory as long as a page that is used heavily. It suffers from Be lady's anomaly.

Example: Suppose the algorithm is FIFO and there is only one page frame. A tick (✓ **)** indicates a page fault.

- **1-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig 7: FIFO representation for 1 frame

In a one-page memory: **20 faults**.

Indeed, since there is only one page frame, and the same page is never referenced twice in a row, each reference produces a fault.

- **2-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	1	1	0	0	0	4	4	3	3	3	2	2	2	0	0	0	0	1
	0	0	2	2	3	3	3	2	2	0	0	0	1	1	1	1	7	7	7
	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓		✓		✓		✓

Fig 8: FIFO representation for 2 frames

In a two-page memory: **15 faults.**

- **3-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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
	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓			✓	✓			✓	✓	✓

Fig 9: FIFO representation for 3 frames

In a three-page memory: **15 faults**

- **4-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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	0	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
	✓	✓	✓	✓		✓		✓			✓			✓	✓			✓		

Fig 10: FIFO representation for 4 frames

In a four-page memory: **10 faults**.

- **5-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	7	7	7	7	7	4	4	4	4	4	4	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	7	7
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1
					3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	✓	✓	✓	✓		✓		✓										✓	✓	✓

Fig 11: FIFO representation for 5 frames

In a five-page memory: **9 faults**.

1.2 Optimal page replacement (OPT): Replace the page that will never be needed again, or at least will be needed furthest in the future.

- **1-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig 12: Optimal representation for 1 frame

In a one-page memory: **20 faults**.

- **2-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	1	2	2	3	3	3	3	3	3	3	2	2	2	0	0	0	0	1
	0	0	0	0	0	0	4	2	2	0	0	0	1	1	1	1	7	7	7
✓	✓	✓	✓		✓		✓	✓		✓			✓	✓		✓		✓	

Fig 13: Optimal representation for 2 frames

In a two-page memory: **13 faults**.

- **3-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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	0	4	4	4	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
✓	✓	✓	✓		✓		✓			✓			✓				✓		

Fig 14: Optimal representation for 3 frames

In a three-page memory: **9 faults**

- **4-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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
✓	✓	✓	✓		✓		✓						✓				✓		

Fig 15: Optimal representation for 4 frames

In a four-page memory: **8 faults**.

- **5-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

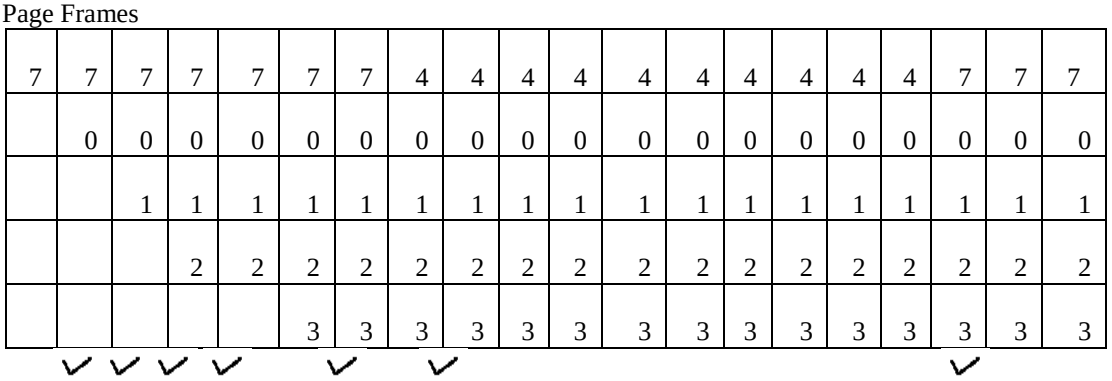


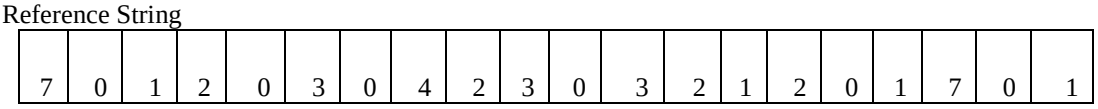
Fig 16: Optimal representation for 5 frames

In a five-page memory: **7 faults**.

1.3 The LRU algorithm: Replace the page that was *least recently used.* Based on the assumption that the recent past can be used to predict the near future.

- **1-Frame**

Reference String



Page Frames

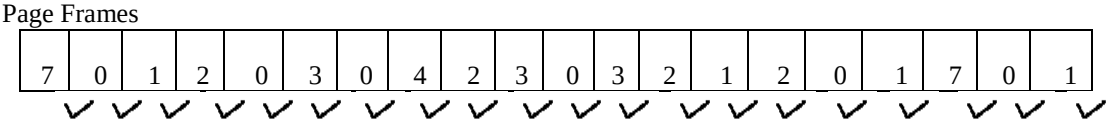
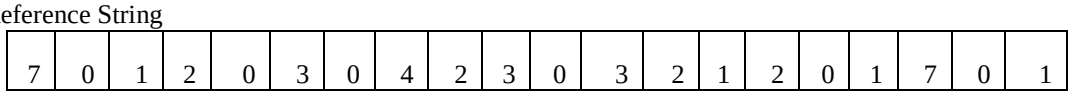


Fig 17: LRU representation for 1 frame

In a one-page memory: **20 faults**.

- **2-Frame**

Reference String



Page Frames

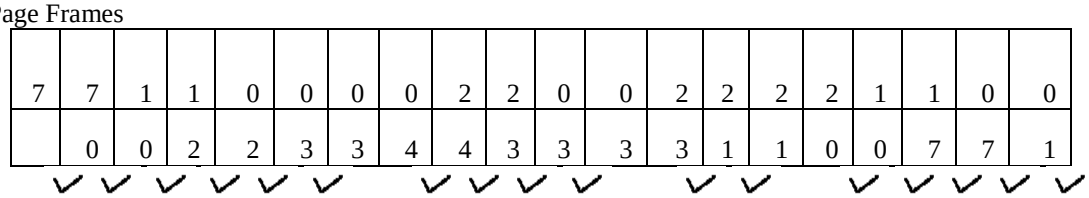
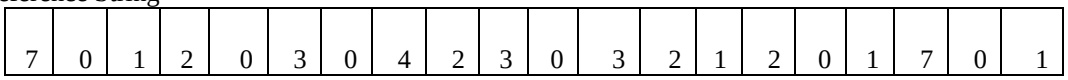


Fig 18: LRU representation for 2 frames

In a two-page memory: 17 **faults**

- **3-Frame**

Reference String



Page Frames

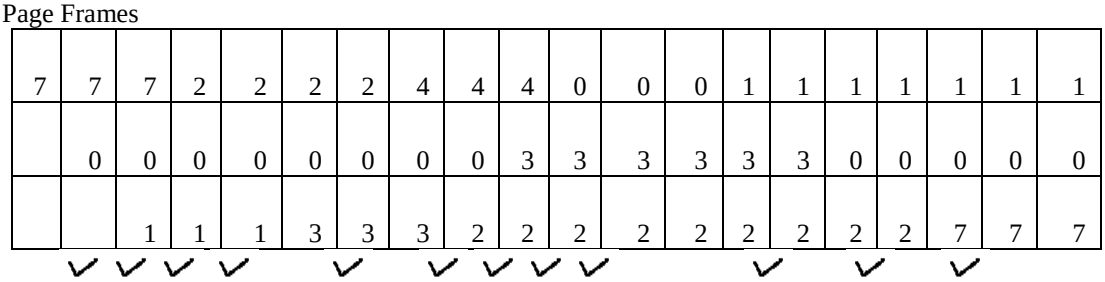


Fig 19: LRU representation for 3 frames

In a three-page memory: **12 faults**.

- **4-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

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
✓	✓	✓	✓		✓		✓							✓			✓		

Fig 20: LRU representation for 4 frames

In a four-page memory: **8 faults**.

- **5-Frame**

Reference String

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Page Frames

7	7	7	7	7	7	7	4	4	4	4	4	4	4	4	4	4	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	1	1	1	1	1	1	1	1	1	1	1	1	1
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
					3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
✓	✓	✓	✓		✓		✓										✓		

Fig 21: LRU representation for 5 frames

In a five-page memory: **7 faults**

V. EXPERIMENTAL RESULTS

In order to assess the proposed work (with respect to page faults and hit ratio) and comparing it with the other algorithms the program implemented in C++ language and have been simulated using MATLAB. In this experiment, hit ratio is reasoned as the compare standard. For this purpose, 20 random sequences with length 20 were imposed to FIFO, Optimal, LRU each with five cases for memory frame number, include 1-frame, 2-frame , 3- frame, 4- frame, 5- frame were reasoned and outcomes have been reported in Table.1 Based on this table, the bar diagram for these results can be shown as chart 1and in chart 2.

Page Replacement Technique		**Number of Frames**				
		1-frame	2-frame	3-frame	4-frame	5-frame
FIFO	Page Fault	20	15	15	10	9
	Hit Ratio	0%	25%	25%	50%	45%
LRU	Page Fault	20	17	12	8	7
	Hit Ratio	0%	15%	40%	60%	65%
OPTIMAL	Page Fault	20	13	9	8	7
	Hit Ratio	0%	35%	55%	60%	65%

Table 1: Comparison of Page Replacement algorithms

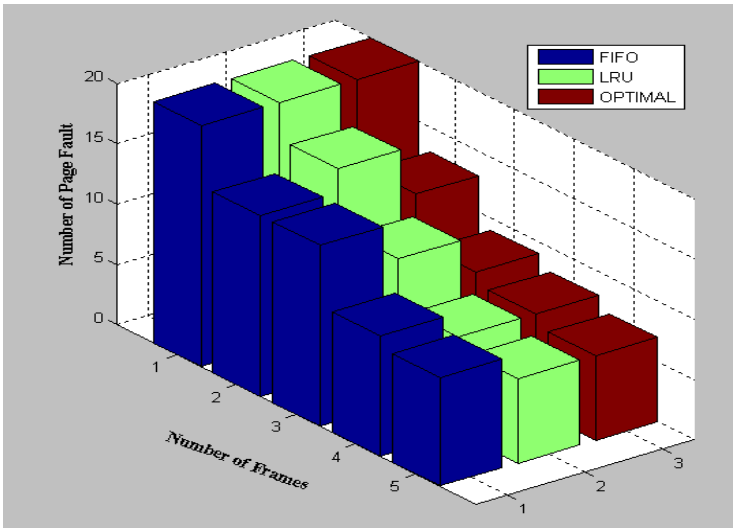


Chart 1: Bar graph representing comparison between FIFO, OPT and LRU on the basis of Page fault rate

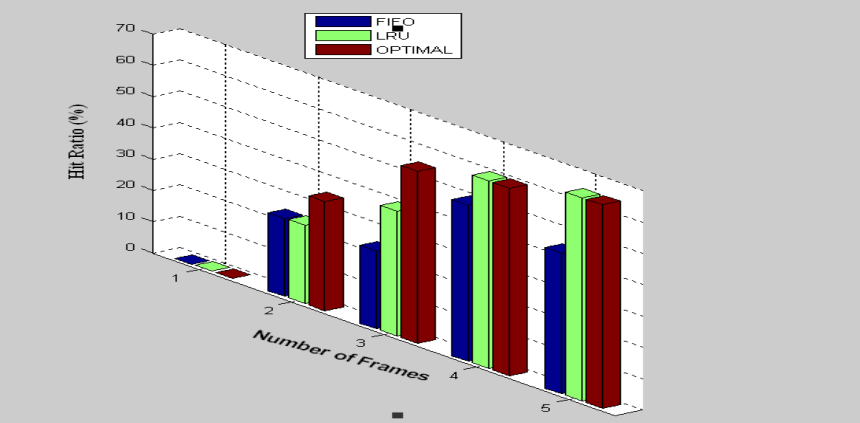


Chart 2: Bar graph representing comparison between FIFO, OPT and LRU on the basis of hit ratio

VI. CONCLUSION

Studying the page replacement algorithms has suggested that the LRU has had the best results among all practical algorithms, Optimal used as a benchmark and FIFO suffers from an anomaly (Be lady's anomaly): sometimes increasing the size of memory increases the fault rate. In this paper different page replacement algorithm are studied and compare with respect to page faults and hit ratio and comparing it with the each other, the algorithms are implemented in C++ and they have been simulated using MATLAB.

REFERENCES

[1] 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.

[2] Heikki Paajanen, "Page replacement in operating system memory management" Master's Thesis, University of Jyväskylä, October 23, 2007

[3] S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy, G Amjad Khan., "A Throghput Analysis on page replacement algorithm",International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, Vol. 2, Issue 2, pp.126-13, Mar-Apr 2012.

[4] Ali Khosrozadeh, Sanaz Pashmforoush, Abolfazl Akbari, Maryam Bagheri, Neda Beikmahdavi., "Presenting a Novel Page Replacement Algorithm Based on LRU" , Journal of Basic and Applied Scientific Research , 2(10)10377-10383, 2012.

[5] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Computers, 50(12):1352-1361, 2001.

[6] Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall, 1992
"Page replacement algorithm" Available:http://en.wikipedia.org/wiki/Page_replacement_algorithm

[7] Operatin system tutorial Available on:http://www.tutorialspoint.com/operating_system/os_virtual_memory.htm

