AN INGENIOUS PAGE REPLACEMENT TECHNIQUE

[1]ASHUTOSH GHILDIYAL, [2]KANCHAN BISHT, [3]H.L. MANDORIA, [4]RAJESH SINGH

[1,2]Students, Department of Information Technology
[3]Head, Department of Information Technology
[4]Assistant Professor, Department of Information Technology
Department of Information Technology, College of Technology, GBPUAT, Pantnagar
E-mail: [1]ghildiyal.ashutosh@yahoo.com, [2]kanchanbisht.rocks@gmail.com,[3]hlm.fcs@gbpuat-tech.ac.in,
[4]rajeshsingh@gbpuat-tech.ac.in

Abstract— Virtual memory, a technique that allows the execution of processes that require memory greater than the actual physical memory available, is implemented using demand paging, which in turn is governed by a page replacement algorithm. In this paper, we propose a page replacement algorithm that performs better than the anterior ones. This algorithm is implemented using the total number of references of elements in the reference string, and a buffer frame. We have shown the results for three and four frames that indicate its better performance compared to FIFO, LRU and optimal page replacement algorithms. Our algorithm aims at enhancing the performance of a system by decreasing the page fault rate and increasing the hit percentage.

Keywords— Virtual memory, demand paging, page replacement, FIFO, Optimal, LRU, Kashu algorithm, page fault, hit percentage.

I. INTRODUCTION

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be at least partially in main memory during execution. To improve both the utilization of the CPU and the speed of its response to users, a general-purpose computer must keep several processes in memory.

Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This technique frees programmers from the concerns of memory-storage limitations. Virtual memory also allows processes to share files easily and to implement shared memory. In addition, it provides an efficient mechanism for process creation. Virtual memory is not easy to implement, however, and may substantially decrease performance if it is used carelessly.

Virtual memory is implemented in systems using Demand Paging. With demand paging the entire process is not loaded into the memory for execution. Frames are loaded as per their requirement. The unused frames are never loaded at all. With demand paging, we can run more processes at a time than allowed by the physical memory available.

Any process that runs on the system is divided into pages. While execution these pages are referenced in a specific order. Page replacement algorithms play an important role in deciding which page to replace when all the frames are occupied and there is no free frame for the referenced page.It is necessary for the continued execution of the processes.

II. EXISTING ALGORITHMS

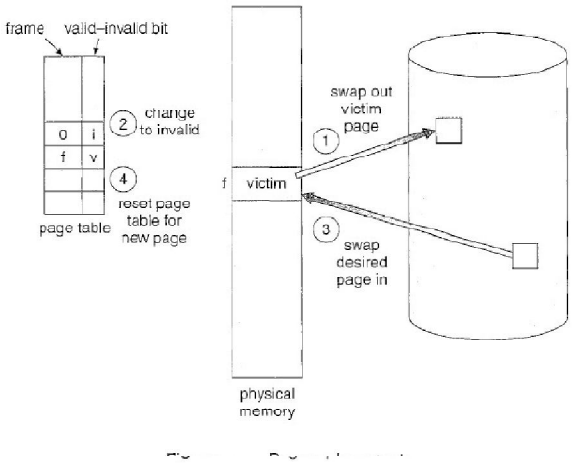
Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d). The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

Secondary memory holds the pages that are not present in the main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for swapping purpose is known as swap space.

Page replacement takes the following approach. If no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory. We can now use the freed frame to hold the page for which the process faulted. We modify the page-fault service routine to include page replacement:

- Find the location of the desired page on the disk.
- Find a free frame:
  a. If there is a free frame, use it.
  b. If there is no free frame, use a page replacement algorithm to select a victim frame.
  c. Write the victim frame to the disk; change the page and frame tables accordingly.
- Read the desired page into the newly freed frame; change the page and frame tables.
- Restart the user process.

Notice that, if no frames are free, *two* page transfers (one out and one in) are required. This situation effectively doubles the page-fault service time andincreases the effective access time accordingly.



Three of the anterior algorithms are discussed next.

2.1. FIFO

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. It suffers from Belady's Anomaly.

2.2. Optimal page replacement

In this algorithm, the page that will not be usedfor the longest period of time is replaced. It does not suffer from Belady's Anomaly.

2.3. LRU

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Like optimal replacement, LRU replacement does not suffer from Belady's Anomaly.

III. OUR APPROACH (KASHU PAGE REPLACEMENT)

This algorithm is implemented using the total number of references of elements in the referencestring, and a buffer frame.For example, if we trace a particular process, we might record the following
address sequence:
0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105
At 100 bytes per page, this sequence is reduced to the following reference string:

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

There are three elements in the reference string- 1,4,and 6. The corresponding total number of references(TNR) for these elements are 6,1,and 4 respectively.
For every reference string these values will be quickly calculated and used to form an efficacious page replacement algorithm.
We use a buffer frame that stores the evicted page. Any reference to this page too would lead in a hit. Thus, when a page is referenced, it is first looked for in the set of frames available, then in the frame buffer, and if not found a page fault is said to have occurred. To deal with this, a page residing in the available set of frames is chosen for replacement according to the following algorithm.

3.1. Algorithm

- Calculate TNR for all unique elements of the reference string.
- Initially, the buffer frame is empty.
- The victim page is the one with the minimum value of TNR.
- If two or more elements have the same value of TNR (which is minimum amongst all the TNR values), then, the victim page is selected according to LRU. If none of the pages have been used then the selection is done as per FIFO.
- The victim page is written onto the frame buffer. If the frame buffer is not empty then its contents are overwritten. The page and frame tables are changed accordingly.
- The desired page is brought onto the memory storage and written onto the cleared frame. The page and frame tables are changed accordingly.

3.2. Experimental Results

To demonstrate our work, we have used the following reference string:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
The following table contains the total number of references for various elements.

Element	*TNR Value*
7	2
0	6
1	4
2	4
3	3
4	1

For three frames:

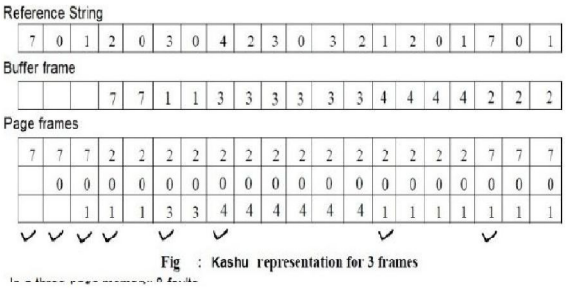


Fig : Kashu representation for 3 frames

For four frames:

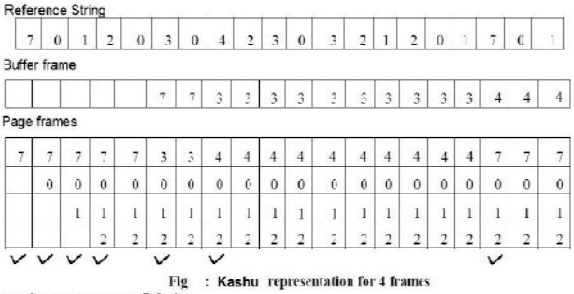


Fig : Kashu representation for 4 frames

n a four-page memory: 7 faults

3.3. Comparison with anterior techniques

FIFO-3 Frames

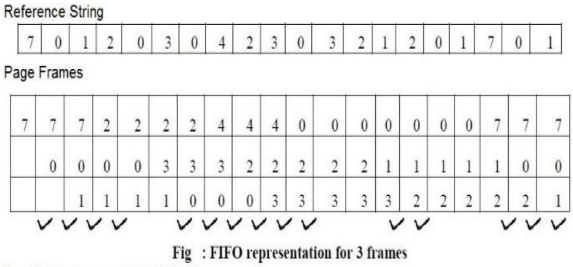


Fig : FIFO representation for 3 frames

In a three-page memory: 15 faults

FIFO-4 Frames

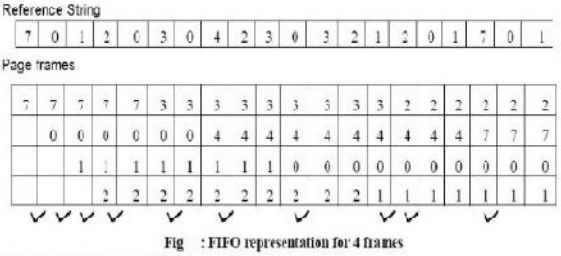


Fig : FIFO representation for 4 frames

In a four-page memory: 10 faults

LRU-3 Frames

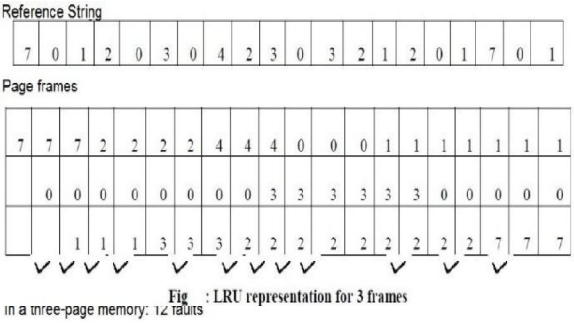


Fig : LRU representation for 3 frames

In a three-page memory: 12 faults

LRU-4 Frames

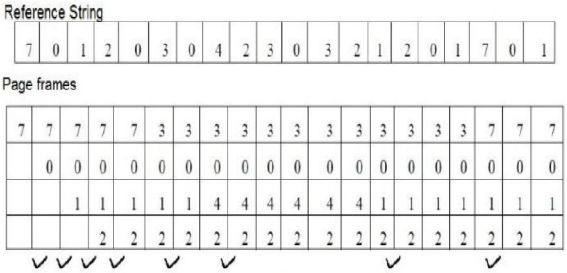


Fig : LRU representation for 4 frames

OPT-3 Frames

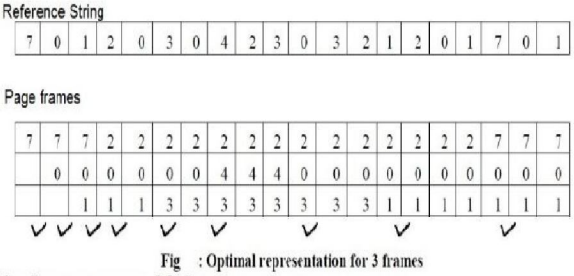


Fig : Optimal representation for 3 frames

OPT-4 Frames

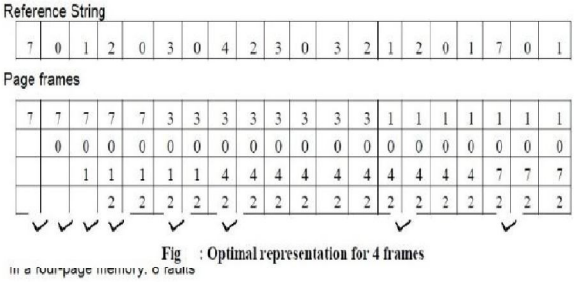


Fig : Optimal representation for 4 frames

In a four-page memory: 8 faults

IV. ADVANTAGES

- It provides a high hit percentage thereby increasing the efficiency of the system.
- Fault rate is minimum as compared to anterior algorithms.
- It does not suffer from Belady's anomaly.

CONCLUSION

The Kashu page replacement algorithm is an efficacious algorithm that reduces the number of page faults and hence increases the hit percentage. This in turn increases the overall efficiency and performance of the system. The decision for page replacement which is made on the basis of number of references, makes the algorithm efficient, and this efficiency is further enhanced by the usage of a buffer frame.

The page faults incurred using Kashu algorithm for an n-page memory system is always less than or equal to those encountered when we use anterior algorithms with a (n+1)-page memory system. Thus, the proposed algorithm promises high speed laced with high performance of the system.

REFERENCES

[1] Silbershatz, Galvin, Gagne, Operating System Concepts, Eight Edition.

[2] William Stallings, Operating Systems, Sixth Edition.

[3] Andrew S. Tanenbaum, Operating Systems Design and Implementation, Third Edition.

[4] Wang Hong, "Study of Page Replacement Algorithm based on Experiment", International conference on Mechanical Engineering and Automation, Advances in Biomedical Engineering, volume 10, 2012.

[5] Ali Khosrozadeh, Sanaz Pashmforoush, "Presenting a Novel Page Replacement Algorithm based on LRU", Journal of Basic and Applied Scientific Research, 2012.

[6] Pooja Khulbe, Shruti Pant, "Hybrid (LRU) page Replacement Algorithm", IJCA, Volume 91, April 2014.

[7] Yogesh Niranjan and Shailendra Tiwari, "Design and Implementation of Page Replacement Algorithm for Web Proxy Caching", IJCTA, Volume 4, April 2013.

[8] Anupam Bhattacharjee, Biolab Kumar Debnath, "A New Web Cache Replacement Algorithm", IEEE Conference on Communications, Computers and Signal Processing, August, 2005.

[9] 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.

[10] 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.