Comparative Study of Page Replacement Algorithms
Sushma Grewal and Manisha Agarwal
Banasthali Vidyapith University, Tonk, Rajasthan.

ABSTRACT
A page fault is a kind of delay, put forward by the system hardware on accessing of a page by a process in execution, which is plotted in the virtual memory, but not really stacked to the main memory. The set up which observes the interrupt is known as the CPU’s memory management unit (MMU), whereas the exception handling program which tackles these interrupts is mainly a section of systems program kernel. During managing an interrupt, the systems program normally take attempts in making the needed page obtainable at the locale in physical memory, or aborts the process in a scenario of unauthorized memory retrieval. In this paper we have done a survey on different techniques to handling a page fault or an interrupt.

© 2016 Elixir all rights reserved.

Introduction
Virtual memory is an attribute of a systems program which authorizes a system to recompense for inadequacy of physical memory through the transfer of information pages from arbitrary retrieval to disk storage on a temporary basis [2]. When a page demanded by a process seems unavailable in the memory, it is termed as a page fault or an interrupt. This generally leads to the closing of the demand. A page is an unalterable size memory section that is utilized in form of a unit for transfer betwixt physical memory and an extrinsic repository. A page fault happens if a process approaches a page which is plotted in the address space, but not stacked into the physical memory.

Page fault handling is generally the job of kernel. If a process tries to approach a page which is presently not plotted to a locale in main memory, the kernel acknowledges by plotting that page to the locale and, if required, nourishing the page by including the information from secondary store. If that information is unavailable in a locale which is smoothly approachable through the kernel, then, the onus of managing the interrupt is deployed to the user space.

One case when user-space page fault managing can be practical is for the live moving of virtual systems from one physical compeer to the other. Moving can be finished by aborting the system, replicating its entire address space to the novel compeer, and restoring the system in that place. But perhaps address area is huge and occasionally utilized; replicating an entire address area leads to a plenty of inessential effort and a observable halt ahead the moves machine restores. Alternatively the virtual system’s address area could be request-paged from the antiquated compeer to the novel one, it can restore much rapidly and the replicating of unutilized information can be ignored.

The following are the steps taken by systems program to handle a page fault:
1) Hardware confines to kernel mode.
2) Program counter is stored in the stack.
3) CPU registers stores the current instruction state.

4) For saving general registers and other information, an assembly code program is executed.
5) A page fault is founded by the operating system.
6) It finds a virtual address which caused the page fault by means of using a hardware register or by using the software.
7) Checking of virtual address is done for ensuring that it is valid or not. If it is not valid then process is killed.
8) It is also checked to see if a protection fault has occurred.
9) If virtual address is valid and if protection fault has not occurred, then, operating system checks for a free frame.
10) If there is no frame free, then page replacement algorithm is run.
11) If selected page frame founds to be dirty, page is scheduled for transfer to disk.
12) Context switching happens and the process which caused the fault is suspended and another process is run.
13) Frame is marked as busy.
14) The page is loaded from the disk, if and only if the page frame is clean.
15) When the page has been loaded, page frame is set in normal state.
16) Instruction that caused the fault is set to initial state, program counter is reset to point this instruction.
17) Scheduling of the program that caused the fault, is done.

Fig 1. Page Fault Handling [5]
18) Operating system runs Assembly language program is executed by the operating system for reloading the registers and other state information.

19) Control is shifted back to user space for execution.

To load the pages in the physical memory, demand paging is required.

If there survives many more processes than can stay in the memory. So to free a frame from the physical memory to load required or demanded page, we requires a page replacement algorithm.

The lesser the time taken by the processor in managing programs, the more its speed and it becomes more effectual. For this purpose the following page replacement algorithms can be used:

1. **FIFO page replacement (First-in-first-out):**
   This algorithm is simple to grasp and implement. But it is not effectual every time in terms of performance. The first-in-first-out (FIFO) algorithm handles the page frames allotted to a job as a disc shaped buffer, and pages are eliminated in round-robin manner. The main requirement is of a pointer which revolves via page frames of the job. This is therefore one of the easiest page replacement algorithms to execute.

   The reason for this verdict, other than its coherence, is eliminating the page which has been in memory the longest:
   A page delivered into memory a long time before may have now fallen out of use. This conclusion will often be mistaken, since there will often be sections of process or information which are strongly utilized through the life of a process. Those pages will be frequently paged in and out by the FIFO algorithm. But it has belady’s anomaly. This is the term used when the number of page frames increased leads to the increment in number of page faults for a stated memory retrieval style. It can be clearly understood by the following illustration.

   Given page reference stream: 7 4 2 4 7 6 4 8 4 6 8 2
   ![FIFO Page Replacement Diagram](image)

   The number of crosses denotes the number of page faults. Therefore in this case the number of page faults is 9.

   Since this algorithm possess belady’s anomaly, LRU can be used to avoid this condition.

2. **Optimal page replacement**
   The foremost page replacement algorithm is simple to illustrate but impractical to execute. When the event of page fault happens, a few deposits of pages are in the memory. One out of these pages is recommended on the just following command. The remaining pages are not recommended unless 10,100, may be 1000 commands later. Every page can be marked with the number of commands which will be implemented ahead that page is first recommended.

   This algorithm states that page having the largest mark should be eliminated. This algorithm has an issue that it is impractical to utilize. During the page fault event, the systems program has not a method of suggesting when every page will be recommended upcoming. This can be more clear with the help of the following illustration.

   Given page reference stream: 7 4 2 4 7 5 4 7 6 4 5 6 4
   ![Optimal Page Replacement Diagram](image)

   The crosses denote the number of page faults. Here, the numbers of page faults are: 9

   ![Optimal Page Replacement Diagram](image)

   The crosses denote the number of page faults. In this case, the number of page faults is: 5

   3. **Least Recently Used page replacement:**
   A good estimation to the optimal algorithm is founded on review that pages which are strongly utilized in some previous commands will mostly utilized in some upcoming commands. Oppositely pages which have not utilized for a long time will possibly remain unutilized for a long duration. This design propose a practical algorithm: when a page fault happens, suggest the page which is not utilized for the longest duration. This technique is commonly known as least recently used (LRU) page replacement.

   To execute this algorithm, it is required to manage a stack of all pages in the memory, with the most recently utilized page at front and least recently utilized page at the bottom position and therefore removing the bottom frame page. Thus we have a free frame to load the demanded page. This can be easily clear with the help of the following illustration.

   Given page reference stream: 2 3 4 5 3 6 3 1 5 6 3 6 5
   ![LRU Page Replacement Diagram](image)

   The crosses denote the number of page faults. Here, the numbers of page faults are: 19
Conclusion

We have studied three different kinds of page replacement algorithms. After a review on these, we can conclude that least recently used (LRU) page replacement algorithm is the best one to use. Although FIFO is easy to implement, but it posesses belady’s anomaly that can be eliminated by using LRU. Also, optimal is easy to understand but impractical to execute since it is based on future prediction. The following shows the hit ratio on increment of cache size for all the above three page replacement algorithms.

References