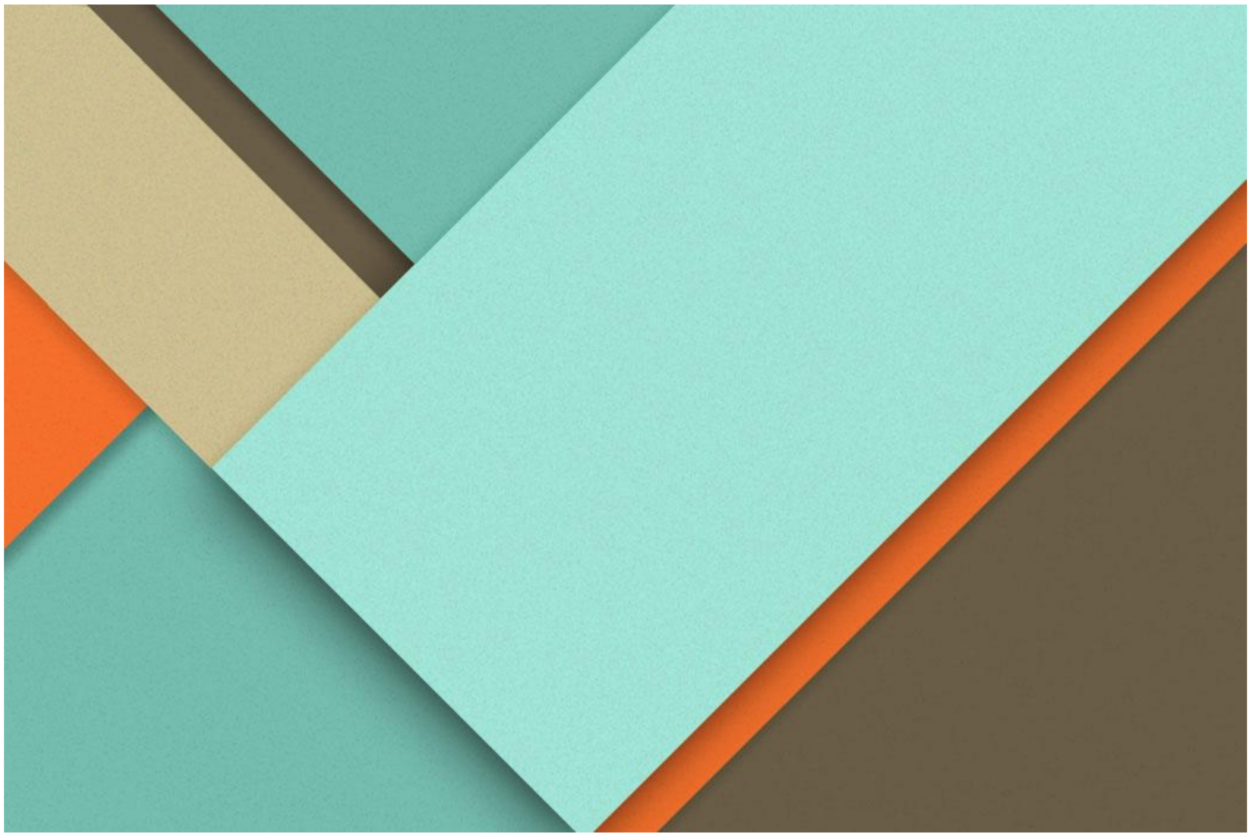

Virtual Memory

23.03.2023

—

Ronit Chinda
2101174

Overview

Virtual Memory is an essential part of the memory management system. Hence it is important to understand the complex internal working of the virtual memory. We have used Basic code given by University of Notre-Dame [1], [2] for simulating virtual memory in the user environment. This has allowed us to work with high level programming language rather than working at the kernel level. We have implemented page fault handling in an incremental manner. Further we have implemented three page replacement algorithms. Finally, we have compared the performance of the page replacement algorithms to get a better understanding of the working of the system.

INTRODUCTION

A computer programmer assumes that their program will have access to the full physical memory (this of course will not happen). Thus, the programs which have to run on the system will almost certainly end up being much larger than what the RAM can hold at a time. Virtual memory allows us to run programmes much larger than the size of physical memory.

Thus, in a Virtual memory system only parts of a code are available in RAM (Main Memory) at any point of time with the rest stored in the non-volatile secondary storage or disk. The locality of references ensures that this is a reasonable thing to do. Most of the time, the things which are required are present in the physical memory. The programs are thus divided into chunks called ’pages’. Once in a while however, a page which is requested is not available and thus has to be fetched from the secondary memory. This is called a ’page fault’. So, Operating Systems effectively give the programs an expanded address space to work in with the OS having to take the responsibility of handling the added complexity involved. This is accomplished using data structures called page tables which map the virtual addresses to their actual physical storages (page frames).

Page replacement algorithms

We implemented the following page replacement algorithms:

Random page replacement- This follows the principle, ”if you don’t have much idea about a process just do things randomly and hope that the problem[here page fault] will occur rarely” i.e. if a page fault occurs and a page is to be replaced, then randomly select a page (frame) from the page table and replace it. Frankly speaking this is not a very sophisticated algorithm. It might throw out the most heavily used page too. But we hope that if the number of pages is pages large enough, then it will perform okay on an average. Though it will certainly not be optimal.

First in first out (FIFO)- It assumes that what is the oldest page must be the one which is not used recently. Hence it kicks out the page which came first i.e. the oldest page from the page table. However, this approach may not be correct at all because an old page may also be the one which is used frequently. Therefore, if we replace that page, then it will make the page management inefficient. Thus, we expect this algorithm to perform poorly.

Least Recently Used (LRU)- This algorithm also considers the time of page usage. That is,whenever a page is accessed, it is marked as most recently used. This is based on the locality of reference in time - if a page is referenced now then there is more probability of it being referenced again in the near future rather than a page which was referenced in the past.We made some major changes in the basic code to implement LRU. This is because the code was mainly for allowing us to implement basic PRAs which would work during the page faults. But LRU requires updating the page list whenever the page is accessed, not just when there is a page fault.

1. We have used the linked list approach to implement LRU wherein each page is a node in the page list. Whenever a page fault occurs,if the page table has an empty frame then a new node for that page is added in the linked list.
2. If we are required to replace a page, then the page at the head of the list is the one which is least recently used and is replaced. The new page is stored at the tail of the list indicating that it is the most recently used.
3. Finally if no fault occurs, i.e if the page is already in the linked list, then it is found out and shifted to the tail of the list again indicating that it is the most recently used.

Memory Management Method :

Scan refers to the process of scanning the pages in memory to identify which ones are in use and which ones can be freed up. This is done by the operating system's memory manager, which keeps track of which pages are currently in use and which are not.

Sort refers to the process of arranging the pages in memory in a particular order. This can help to optimize memory usage and reduce the number of page faults (when the operating system needs to retrieve a page from secondary storage because it is not currently in RAM).

Focus refers to the process of prioritizing certain pages in memory over others. This is done by the operating system's memory manager, which may give higher priority to pages that are currently being accessed by the CPU or that are part of a currently running program.

Observations

No of Pages : **10**

No of Frames : **10**

	Random	FIFO	Custom
Page Faults	20	20	20
Disk Reads	10	10	10
Disk Writes	0	0	0

No of Pages : **12**

No of Frames : **7**

	Random	FIFO	Custom
Page Faults	84	94	85
Disk Reads	48	57	48
Disk Writes	34	37	37

No of Pages : **100**

No of Frames : **10**

	Random	FIFO	Custom
Page Faults	1730	1620	1634
Disk Reads	1081	1010	1015
Disk Writes	649	619	649

Results

Through this project we have been able to understand the role and working of virtual memory in the memory management system of the OS. This simulation has allowed us to understand the memory mapping between the virtual and the physical memory environments of the system. It has also allowed us to identify different reasons for page faults and how to handle them. Finally we were able to successfully implement three of the page replacement algorithms and compare their performances to some extent. In results, we conclude that LRU always performs either equally well or even better than FIFO.