DESIGNING A VIRTUAL MEMORY MANAGER

ABHINANDHU A
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20002@am.students.amrita.edu

HARIPRASAD S
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20035@am.students.amrita.edu

M MAHADEV
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20045@am.students.amrita.edu

MADDALA H S M KRISHNA KARTHIK
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20046@am.students.amrita.edu

MARASANI JAYASURYA
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20048@am.students.amrita.edu

***Abstract*—The architecture of a virtual memory manager is described in this work, which comprises building a program that converts logical to physical addresses. This program will take logical addresses from an input file, interpret every logical address towards its associated physical address using a TLB (translation lookaside buffer) and a page table, and show the changed logical address with its associated physical address in an output file. The purpose of this study is to model the stages involved in converting logical addresses to physical addresses. It handles page faults as well as transforming logical to physical addresses. After the program is completed, we will report the TLB prediction performance, Page prediction performance, and Page-fault rate.**

***Index Terms*—TLB,FIFO,Logical Address ,Physical Address, Frame Number, Page Number**

I. Introduction

Computer systems have become a vital and inescapable part of our life in today's world. We utilize computers for a variety of purposes and in a variety of ways because they make our jobs simpler and more convenient. Working with computers that don't operate as speedily as we want them to or that can't handle particular operations due to a lack of system resources can be very unpleasant or stressful at times. When system resource restrictions become a severe impediment to obtaining our maximum productivity, we frequently investigate obvious ways to upgrade the system, such as moving to a speedier Central processing unit, increasing additional memory space (RAM), installing utility programs, and so on. System Optimization is the process of ensuring that its operating system makes the best use of its resources. The notion of Virtual Memory was established to make the procedure of system optimization easier.

Virtual Memory is a key feature of the operating system that allows a task utilizing Random - access memory that is fully independent of other processes running at the same time and uses more space than the actual amount of RAM. To create a greater range of contiguous addresses, virtual memory combines current RAM and dormant hard disc. Virtual memory is just a tool for multitasking development. Virtual memory enables the creation of programs that behave like directly addressable RAM. Virtual memory makes it easy to specify each application program by masking main memory fragmentation or accessing memory using relative addressing.

.

Virtual memory is a broad notion that encompasses memory virtualization in general. Virtual memory was designed to help an extension as simple to use as feasible for consumers, rather than to enlarge RAM. Virtual memory is a type of memory that is designed to regulate the transfer of data and programs among secondary memory and Random - access memory to create the illusion of a single huge store. This strategy is used to make the programmer's job easier when the program code size surpasses the primary memory size. External fragmentation is eliminated, and internal fragmentation is minimized, thanks to virtual memory.

II. Literature Review

The fundamental need for flash storage is being known as mobile gadgets and embedded systems, according to Liang Shi et al. [1]. Flash memory is a tiny and lightweight storage device with low energy consumption and stress resilience, making it an ideal option to replace traditional hard discs. Flash memory emphasizes asymmetric read and write speeds. Erase and write operations on flash memory are intertwined. Conventional leadership techniques and disk space are built based on hard disc as a storage system in Flash ram-based systems to lower the number of write processes and increase input/output performance.

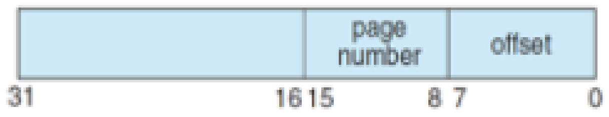
Flash memory may dramatically lower the cost of page swapping, making it a good device to employ as a backup in virtual memory. Read, write, and erase commands are supported by flash memory. If a flash memory page has already been written, it cannot be rewritten, and the matching block must be wiped before data may be written to it. These limitations are known as the Seunggu Ji et al. proposed "erase-before-write" requirement. [2] A data segmented program and its code are restructured in virtual memory to lower the implementation complexity of a program; this approach is known as program restructuring.

Virtual memory is a hierarchical organization made up of small, quick primary memory and vast, slow secondary memory that is used to expand the capabilities and potential of a computer system. A program's or user application's virtual address space is broken into matching size pages in a paged virtual main memory, and memory location is partitioned into equivalent size pieces called page frames. The fundamental goal of program restructuring is to enhance page memory utilization while cutting the number of pages and the program's space-time execution cost. The information acquired by the interpreter is used to do a static program rearrangement before loading time, as reported by Stephen J. Hartley et al. [3].

The performance of a page-based virtual memory process is affected by the frequency of page faults, according to Steven P.Smith and John Kuban et al. [4]. The incalculable effort is now being expended to reduce the detection accuracy of virtual into physical address mapping. Transfers from disc to physical memory are often made in units of pages, and theoretical memory is typically divided into pages of identical size. When a virtual address that is not in physical memory is addressed, a cache replacement algorithm is employed to determine which page in primary memory is replaced by the referenced page. The behavior of simulation references was measured by analyzing APOLLO address traces, which were subsequently collected using I.M.S. (integrated measurement system). The Page replacement algorithm was utilized to utilize the address traces in a virtual memory system. Page length and physical storage capabilities can be easily changed using a software modeling method to see how they affect overall performance.

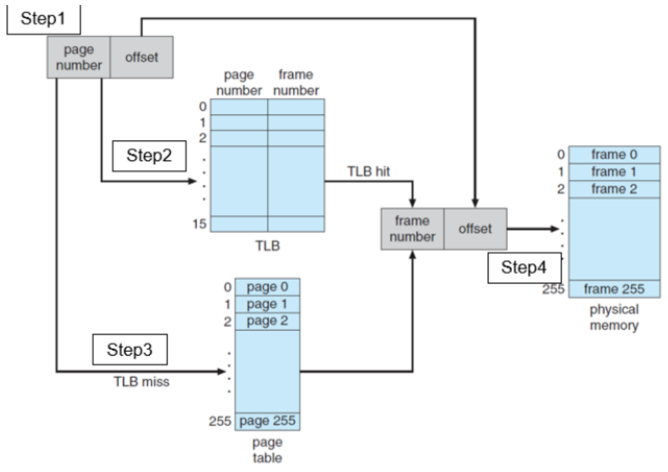
III. PROBLEM DESCRIPTION

The virtual memory manager simulates the process of translating logical to physical addresses. As shown below, the page number and offset are first extracted from the logical address. (Logical address - Page Number + Offset)



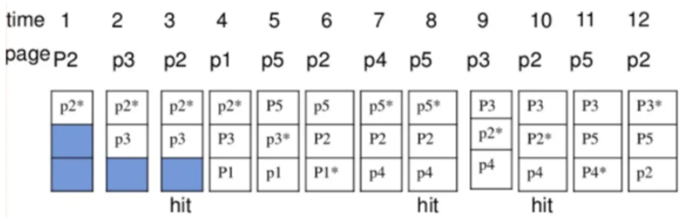
The following formulas are used to obtain the page number and offset out from logical address.
Page Number = Logical address / Page size
Offset = Logical address % Page size
The TLB is then consulted. The frame number was calculated from the TLB in the scenario of a TLB-hit. The page table will be examined in the event of a TLB-miss. Whether the frame number was taken out from the page table or perhaps a page fault occurs in the latter instance. Finally, data can be retrieved from physical memory using frame number and offset. (Physical address - Frame Number + Offset)

The following is the formula for data extraction with frame number and offset.
Physical address = Frame number * Page size + Offset The address-translation mechanism is depicted here.



Address-Translation Mechanism

Since the TLB has 16 entries in this case, a replacement approach should be utilized when the whole TLB is updated. Either a FIFO or LRU strategy is utilized to update the TLB. The FIFO (First In First Out) cache replacement technique is utilized in this paper. The easiest page replacement algorithm is FIFO. In this case, the method maintains a record among all pages in cache in a queue, with the oldest page at the top. Whenever a page requires replacement, the first page in the queue is chosen for replacement. The FIFO algorithm's process is depicted here.


time 1 2 3 4 5 6 7 8 9 10 11 12
page P2 p3 p2 p1 p5 p2 p4 p5 p3 p2 p5 p2
hit
hit
hit

The page table has 256 entries, the page size is 256 bytes, the TLB has 16 entries, the frame size is 256 bytes, the virtual memory manager has 256 frames, and the physical memory is 65,536 nibbles. The programming language chosen was C++. TLB (Translation Lookaside Buffer) Page Table are implemented with an Array, and Physical Memory is presented in integer form. The table update technique is FIFO, and the incoming and outgoing records are txt records (address.txt, output.txt).

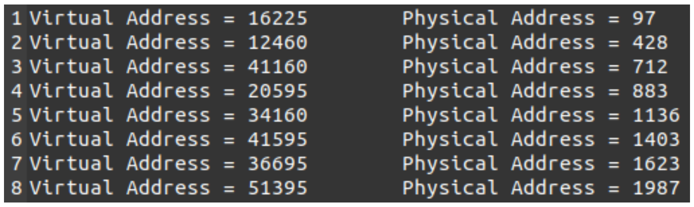
IV. METHODOLOGY

The following are the steps required to create a virtual memory manager that converts logical addresses to physical addresses.

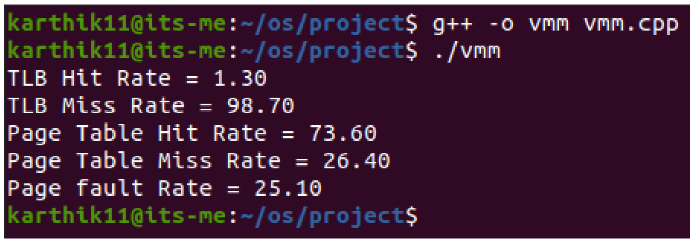
- Read every logical address out from input data at first (address.txt).
- Then, using the equations given earlier, determine Page Number as well as Offset out of each logical address.
- Consult TLB to find the correct frame number based on the page number.
- The frame number gets collected from TLB if TLB is triggered.
- Consult the page table if there is a TLB missing.
- The frame number was calculated if a page table strike occurs.
- If you don't, you'll get a page fault.
- The FIFO page replacement technique is used to address page fault conditions.
- Restart the process after handling the page fault; the frame number could be collected from the TLB.
- Now use the equation given earlier, determine the physical address by appending the frame number and offset.
- Finally, in the output file, provide the virtual address that corresponds to the physical address received (output.txt).
- The TLB prediction performance, Page table prediction performance, and Page fault rate are also printed.

V. Results And Analysis

It has been successfully designed as a virtual memory manager that converts logical addresses to physical addresses. The application reads the file addresses.txt, which includes 1000 logical addresses ranging from 0 to 65535, and converts them to physical addresses. The programme shows logical or virtual addresses being translated into physical addresses. In the output.txt file, it shows the logical address being converted with the appropriate physical addresses. The following are some of the details of the output file.


```
Virtual Address = 16225       Physical Address = 97
Virtual Address = 12460       Physical Address = 428
Virtual Address = 41160       Physical Address = 712
Virtual Address = 20595       Physical Address = 883
Virtual Address = 34160       Physical Address = 1136
Virtual Address = 41595       Physical Address = 1403
Virtual Address = 36695       Physical Address = 1623
Virtual Address = 51395       Physical Address = 1987
```

TLB hit rate, TLB miss rate, Page Table hit rate, Page Table miss rate and Page fault rate are also displayed by the software. TLB hit rate corresponds to the fraction of address references resolved in the TLB, while TLB fault rate corresponds to the fraction of address references not recovered in the TLB. Page Table hit rate corresponds to the fraction of address references resolved in the Page Table, while Page Table fault rate corresponds to the fraction of address references not recovered in the Page Table. The fraction of address referrals that culminated in page faults is known here as page fault rate. The following are the details of the result.



```
karthik11@its-me:~/os/project$ g++ -o vmm vmm.cpp
karthik11@its-me:~/os/project$ ./vmm
TLB Hit Rate = 1.30
TLB Miss Rate = 98.70
Page Table Hit Rate = 73.60
Page Table Miss Rate = 26.40
Page fault Rate = 25.10
karthik11@its-me:~/os/project$
```

Here, TLB hit rate will be low because the logical entries in addresses.txt were chosen randomly and do not represent any memory access locality.

REFERENCES

[1]. YA, Divya. (2019). An Efficient Virtual Memory using Graceful Code. International Journal of Trend in Scientific Research and Development. Volume-3. 623-626.10.31142/ijtsrd23878.

[2].Rafal Kolanski, "A logic for virtual memory ", Electronic Notes in theoretical computer science 217, Pages 61-77, 2008

[3].Bensoussan, R.C.Daley, "The multics virtual memory: concept and design", volume 15, no.5, PP 3078,318,1 May 1972.

[4].X. Zhou, P Petrov, "Towards virtual memory supports in real time and memory constrained embedded applications the interval page system", Received on 11th march 2009, Revised on 19 th march 2009.

[5]. Archana s. Sumant, Pramila M. Chawan, "Virtual memory techniques in 2.6 kernel and challenges", vol 2, no., ISSN : 1793-823; 2, april 2010.

[6].Y. A. Khalidi,, "Virtual Memory Supports for Multiple page Sizes", on pages (104-109), 14-15 oct 1993.

[7].Hung-Wei Tseng, Han-Lin Li, Chia-Lin Yang, "An Energy-Efficient Virtual Memory System with Flash Memory as the Secondary Storage", Low Power Electronics and Design, ISLPED 06, 2006.

VI. APPENDIX

Github Link : https://github.com/karthik1124/Virtual_Memory_Manager