VIRTUAL MEMORY MANAGEMENT SIMULATOR

Submitted By:

JAIDEV CHITTORIA(181IT119)

SIDDHARTH POKHARNA (181IT146)

AYUSH BHANDARI (181IT209)

ABHISHEK KASWAN (181IT201)

IV SEM B.TECH (IT)

Under the guidance of

Dr. B.Neelima

Dept. of IT, NITK Surathkal

in partial fulfilment for the award of the degree

of

Bachelor of Technology

In

Information Technology

at

DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

JUNE 2020

ABSTRACT

Virtual memory management systems are suitable for interactive continuous media applications. Interactive continuous media applications nowadays require large amount of memory for storing code and data and other segments.However traditional operating system might not be able to do so because of ram size less than (4GB) , also they can run out of space if multiple programs are running and sometimes can corrupt data which is accessed by many programs. Since code needs to be in memory to execute but generally entire program is rarely used due to error code,unusual routines and due to large data structures.With the help of virtual memory management we can run partially loaded programs and call resources when required leading to increase in CPU utilization and throughput with no increase in response and turnaround time with less I/O needed to load or swap programs into memory ultimately leading to proper utilisation of CPU resources and faster run of program .In this project we are going to simulate Virtual memory Management based on a text file of logical address with the help of TLB and applying page fault replacement strategies if page fault occurs and displaying physical addresses corresponding to virtual addresses.

TITLE **PAGE NO.**

INTRODUCTION

In this project we are going to make Virtual Memory Management based on a text file input of logical addresses which represents sequential instructions with address range 0 to 2^16-1

This project is the design and implementation of a standalone virtual memory manager, where there is a software-managed TLB. The program is responsible to

- Load a file containing a list of logical addresses
- Translate logical addresses into physical addresses for a virtual address space of size 2^{16} = 65,536 bytes, and
- Output the value of the byte stored at the translated physical address.

PREREQUISITE KNOWLEDGE

- Virtual memory
- Basic terminologies frames,page size,TLB,offset,virtual and physical address
- Replacement strategies for TLB
- makefile and .bin files (basics)
- Concepts of OOPs

OBJECTIVES

The Objectives are :-

To utilize the resources to map memory addresses used by a program, into physical addresses in CPU memory in the following steps

1. Creation of Page Table

2. Creation of TLB and Updating

3. TLB Replacement Strategy

    - LRU
    - FIFO

4. Managing Addresses :-

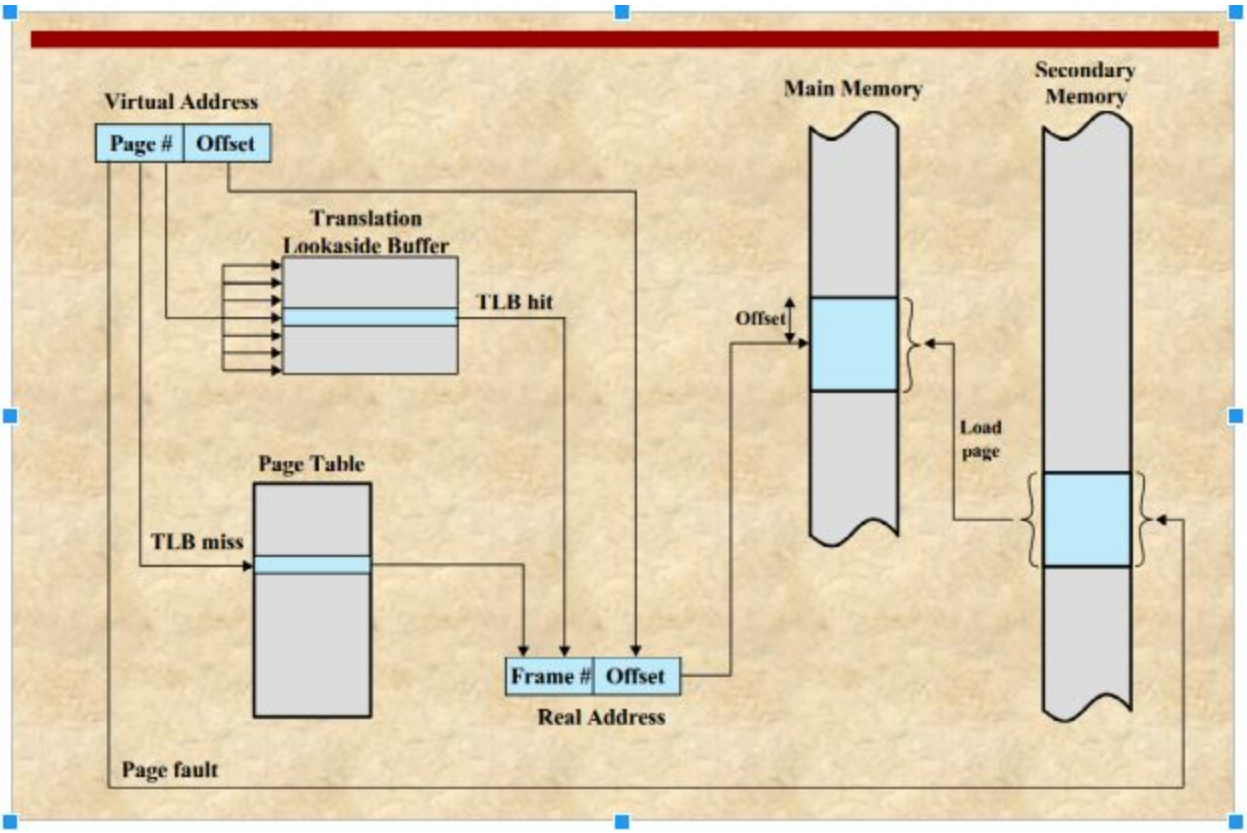
    - Conversion of each logical address to the correct physical address.

    - Extraction of page number and offset from logical address.

    - Finding of frame number associated with page number

    - Using frame number and offset to extract the physical address.
5. Output the virtual as well as physical address and display TLB hits and hit rate and page fault and page fault rate depending upon the TLB replacement strategy chosen

METHODOLOGY



- 16-bit Logical Addresses

  The program reads a file containing a list of 32-bit integer numbers, which represent 32-bit logical addresses. It should be noted that the program only deals with 16-bit addresses. Thus, this simulation implements masking for the rightmost 16 bits of each logical address loaded from the file.

Page No.	Offset
Bits (15-8)	Bits(7-0)

- System Parameters of the Virtual Memory

The page table size is 2^8 bytes; the TLB contains 16 entries. The page size is 2^8^ bytes, which is the same as the frame size. There are a total of 256 frames in the physical memory, meaning that the total physical memory capability is 65,536 bytes (i.e., 256 frames * 256 bytes/frame). The system parameters of the simulated virtual memory is summarized below.

S.No	Type	Size
1	Page Table size	*2^8 bytes*
2	Number of TLB entries	*16*
3	Page size	*2^8 bytes*
4	Frame size	*2^8 bytes*
5	Number of frames	*256*
6	Physical memory size	*65536 bytes*

TIMELINE

No. of Weeks =5

S.No	Week	Checkpoints
1	WEEK1	Understanding of Virtual Memory Management concepts
2	WEEK2	Translation of logical address to physical address
3	WEEK3	Page fault algorithms implementation
4	WEEK4	Updating of TLB
5	WEEK5	Testing and Debugging of code

IMPORTANT FEATURES

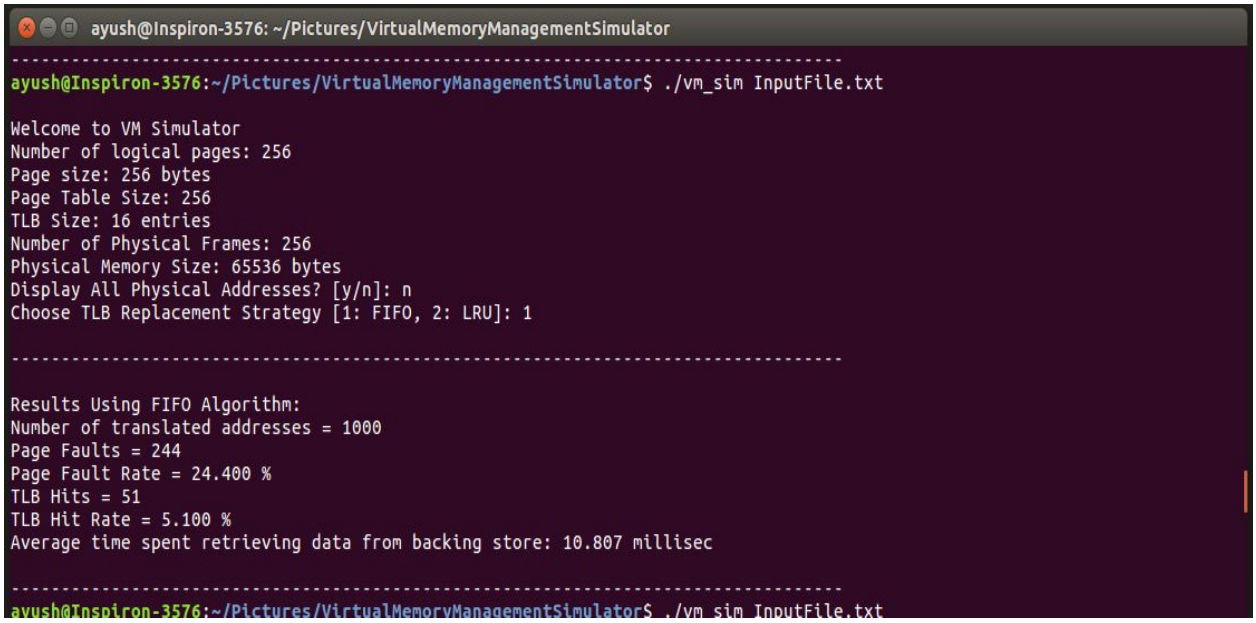
- Users can define the desired size of physical and virtual memory and check the performance of replacement algorithms with average time spent retrieving data from the backing store.
- The solution is quite general and can easily change the different parameters as long as the user is aware of its applications .

```
#define FRAME_SIZE        256
#define TOTAL_FRAME_COUNT 256
#define PAGE_MASK         0xFF00
#define OFFSET_MASK       0xFF
#define SHIFT             8
#define TLB_SIZE          16
#define PAGE_TABLE_SIZE   256
#define MAX_ADDR_LEN      10
#define PAGE_READ_SIZE    256
```

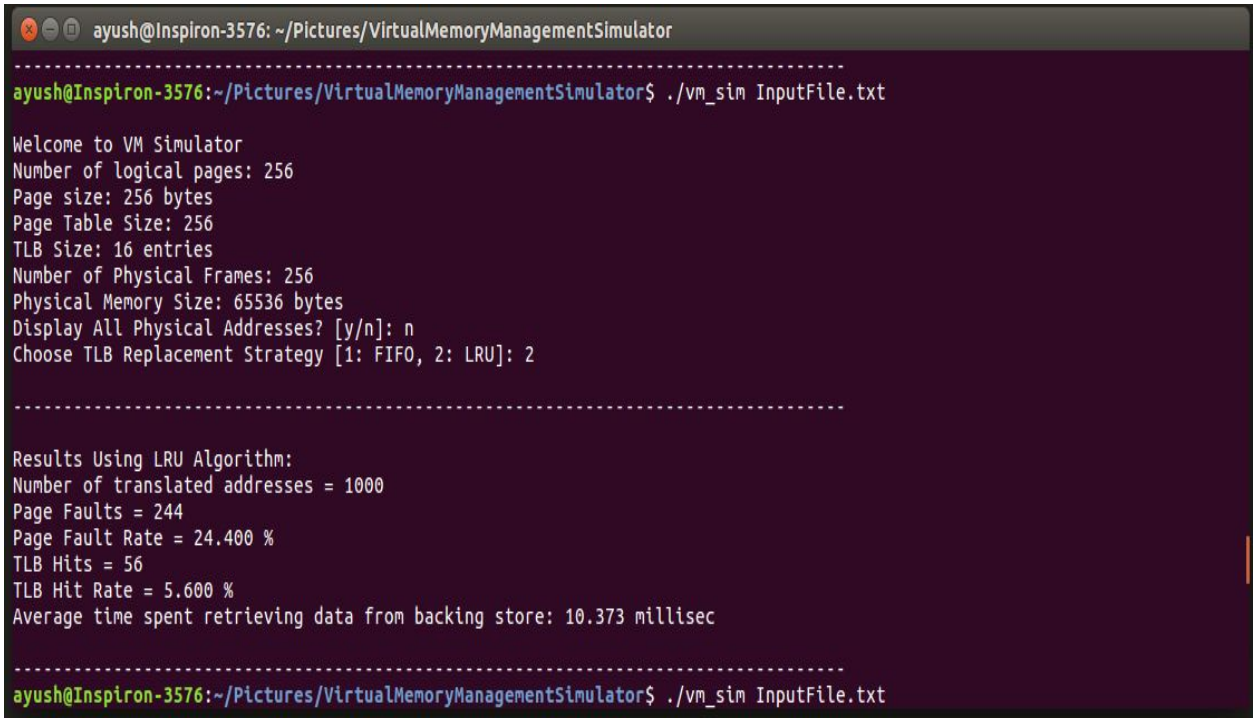
- Users can implement either FIFO or LRU for page fault replacement.

RESULTS

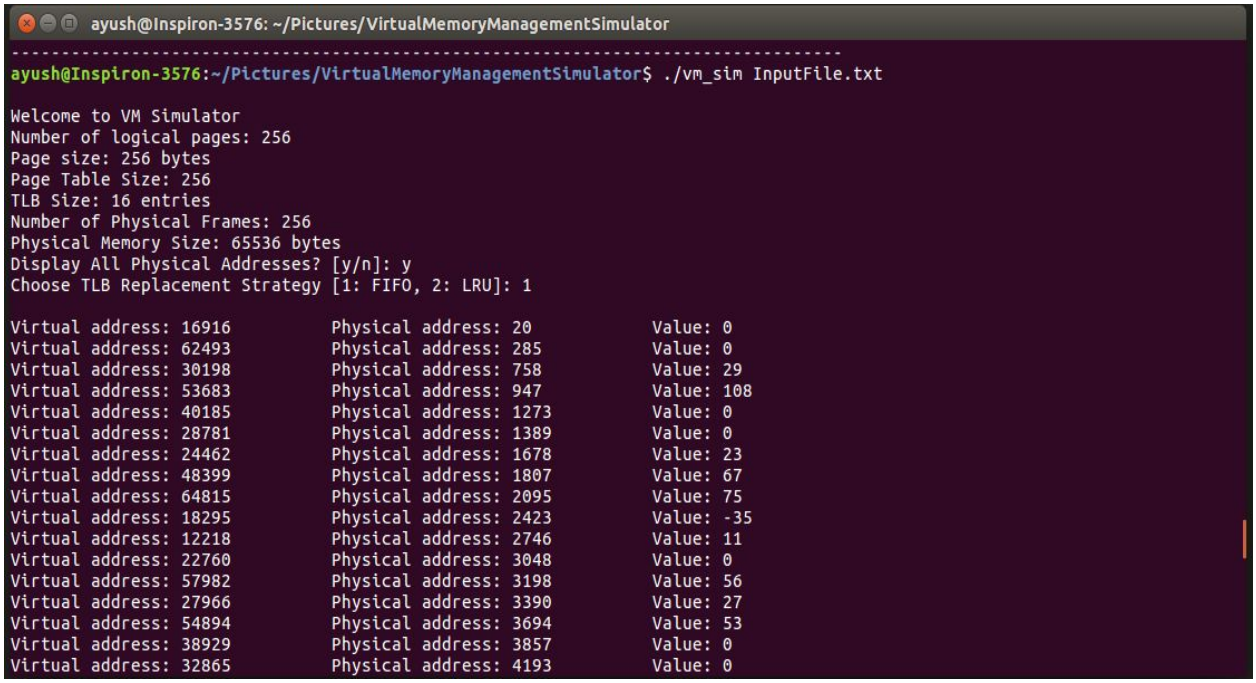
FIFO



LRU



Mapping of Addresses



CHALLENGES FACED

1. A real MMU uses hardware components(like TLB) Which we can simulate by creating a data structure.But This virtual memory simulator uses only algorithmic and software approaches to compute addresses.
2. To guarantee that each logical address is translated to the correct physical address we needed to use bitmasking. So we used a 32-Bit masking function to extract page number and offset number.
3. How to use fgets() function,Proper malloc() error output and exiting, and dynamic memory allocations were some other challenges that we faced.

TEAM LOAD CONTRIBUTION

Jaidev Chittoria	**:**	Creation and updation of TLB
Siddharth Pokharna	**:**	Managing Addresses and input files
Abhishek Kaswan	**:**	FIFO implementation and Page table
Ayush Bhandari	**:**	LRU implementation, testing and debugging of code

CONCLUSION

This simulation uses solely software and algorithmic computations to address an abstraction for storage resources, it does not "realistically" simulate a virtual memory system. Our design has enough space on the page table to accommodate the number of frames in simulated physical memory. It does, however, accurately show how resources are utilized to map memory addresses used by a program, into physical addresses in CPU memory. Also, this implementation allows main storage, as seen by a process or task, to appear as contiguous address space.It also shows how a replacement algorithm works and how data is taken from secondary memory if it is not present in the page table.

REFERENCES

1. Design and implementation of virtual memory-mapped communication on Myrinet (a ieee report by C. Dubnicki ; A. Bilas)
2. Thomas Sterling, Maciej Brodowicz, in High Performance Computing, 2018
3. Dan C. Marinescu, in Cloud Computing (Second Edition), 2018
4. Simulation and verification of the virtual memory management system with MSVL(a iee report by Meng Wang ; Zhenhua Duan ; Cong Tian)
5. Alkassar Eyad, Schirmer Norbert and Starostin Artem, "Formal pervasive verification of a paging mechanism"