Designing a Virtual Memory Manager

Project submitted to the

SRM University – AP, Andhra Pradesh

For the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by:

SWARNAPUDI ISHWAR – AP21110010379

Under the Guidance of

Dr Krishna Prasad

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Certificate

Date: 25-May-23

This is to certify that the work present in this Project entitled "**Designing a Virtual Memory Manager**" has been carried out by **Swarnapudi Ishwar** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr Krishna Prasad

Designation,

Affiliation.

Table of Content

Statement of the problem

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 216 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB as well as a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. The goal behind this project is to simulate the steps involved in translating logical to physical addresses.

Explanation of the problem

Virtual memory is a technique that allows a computer program to access memory addresses that are not physically present in the computer's main memory. This is done by using a page table, which is a data structure that maps virtual addresses to physical addresses. When a program tries to access a memory address that is not currently in main memory, the operating system will first check the page table to see if the address is mapped to a physical address. If the address is not mapped, the operating system will bring the page containing the address into main memory and then update the page table to map the address to the physical address of the page in the main memory.

The project involves designing a virtual memory manager that can translate logical addresses to physical addresses. The virtual address space is 216 = 65,536 bytes. The program will read from a file containing logical addresses and, using a TLB as well as a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address.

The TLB (Translation Lookaside Buffer) is a small, fast memory that is used to store recently used page table entries. When the operating system needs to translate a logical address to a physical address, it first checks the TLB to see if the address is already in the TLB. If the address is in the TLB, the operating system can quickly retrieve the corresponding physical address and continue with the translation process. If the address is not in the TLB, the operating system will need to look up the address in the page table.

The page table is a data structure that maps virtual addresses to physical addresses. The page table is divided into a number of pages, each of which can store a number of page table entries. Each page table entry contains the physical address of the page that contains the data for the corresponding virtual address.

The program will need to use the TLB and the page table to translate logical addresses to physical addresses. The program will first check the TLB to see if the address is already in the TLB. If the address is in the TLB, the program can quickly retrieve the corresponding physical address and continue with the translation process. If the address is not in the TLB, the program will need to look up the address in the page table.

Once the program has found the corresponding physical address, it can then read the value of the byte stored at that address and output the value.

This project aims to simulate the steps involved in translating logical to physical addresses.

Steps/ Algorithm to solve the problem

Step 1: Define the constants and data structures:

Identify the values of PAGE_SIZE, FRAME_SIZE, TLB_SIZE, PAGE_TABLE_SIZE, and PHYSICAL_MEMORY_SIZE. These constants determine the size of various memory components.

Understand the structure of the TLBEntry, which holds a page number and frame number.

Step 2: Understand the helper functions:

getPageNumber(int logicalAddress): Extracts the page number from a given logical address.

getOffset(int logicalAddress): Extracts the offset from a given logical address.

readFromPhysicalMemory(unsigned int physicalAddress): Simulates reading a value from physical memory based on the physical address.

Step 3: Initialize variables and data structures:

Identify and understand the purpose of the variables, such as logicalAddressesFile, physicalAddressesFile, pageTable, tlb, physicalMemory, tlbIndex, tlbHits, pageFaults, and totalAddresses.

Initialize the TLB and page table with initial values.

Step 4: Read logical addresses from a file:

Open the "logical_addresses.txt" file.

Read each logical address from the file.

Extract the page number and offset from the logical address.

Step 5: Handle TLB lookup and page table access:

Check if the page number is present in the TLB.

If found (TLB hit), retrieve the corresponding frame number.

If not found (TLB miss), check the page table.

If the page number is present in the page table, retrieve the corresponding frame number.

If not found (page fault), generate a random frame number and update the page table.

Step 6: Update the TLB:

Update the TLB with the new page number and frame number.

Maintain the order of TLB entries using the tlbIndex.

Step 7: Calculate the physical address and read value:

Calculate the physical address using the frame number and offset.

Call the readFromPhysicalMemory function to retrieve the value from physical memory based on the physical address.

Step 8: Write physical addresses to a file:

Open the "physical_addresses.txt" file.

Write the physical address to the file.

Step 9: Output the result:

Print the logical address, physical address, and value to the console.

Step 10: Output statistics:

Print the number of TLB hits, page faults, and page fault rate to the console.

These steps provide an overview of the code's structure and functionality, allowing you to analyse and understand how it works.

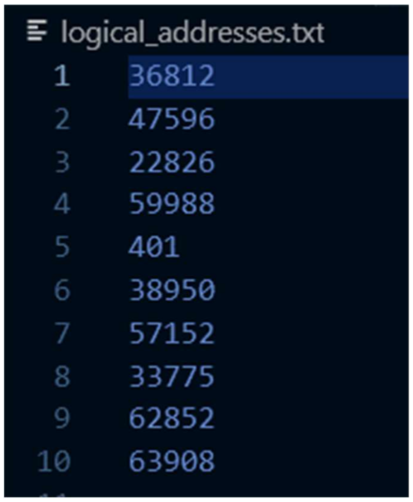
PROGRAM CODE

```
#include <iostream>

#include <fstream>
#include <unordered_map>

// Constants
const int PAGE_SIZE = 256; // Page size in bytes
const int FRAME_SIZE = 256; // Frame size in bytes
const int TLB_SIZE = 16; // TLB size
const int PAGE_TABLE_SIZE = 256; // Page table size
const unsigned int PHYSICAL_MEMORY_SIZE = FRAME_SIZE * PAGE_TABLE_SIZE; //
Physical memory size

// TLB entry structure
struct TLBEntry {
    int pageNumber;
    unsigned int frameNumber; // Changed data type to unsigned int
};

// Function to extract page number from logical address
int getPageNumber(int logicalAddress) {
    return (logicalAddress >> 8) & 0xFF;
}

// Function to extract offset from logical address
int getOffset(int logicalAddress) {
    return logicalAddress & 0xFF;
}

// Function to simulate reading a value from physical memory
int readFromPhysicalMemory(unsigned int physicalAddress) { // Changed data
type to unsigned int
    // Here you can implement the actual reading from physical memory
    // For simplicity, let's assume the physical memory contains random values
    return rand() % 256;
}

int main() {
    // Variables
    std::ifstream logicalAddressesFile("logical_addresses.txt");
    std::ofstream physicalAddressesFile("physical_addresses.txt"); // File to
write physical addresses
```

```cpp
    std::unordered_map<int, unsigned int> pageTable; // Changed data type to
unsigned int
    TLBEntry tlb[TLB_SIZE];
    int physicalMemory[PHYSICAL_MEMORY_SIZE];
    int tlbIndex = 0;
    int tlbHits = 0;
    int pageFaults = 0;
    int totalAddresses = 0;

    // Initialize TLB and page table
    for (int i = 0; i < TLB_SIZE; i++) {
        tlb[i].pageNumber = -1;
        tlb[i].frameNumber = -1;
    }
    for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
        pageTable[i] = -1;
    }

    // Read logical addresses from file
    int logicalAddress;
    while (logicalAddressesFile >> logicalAddress) {
        totalAddresses++;

        // Extract page number and offset
        int pageNumber = getPageNumber(logicalAddress);
        int offset = getOffset(logicalAddress);

        // Check if the page number is present in the TLB
        bool tlbHit = false;
        unsigned int frameNumber = -1; // Changed data type to unsigned int
        for (int i = 0; i < TLB_SIZE; i++) {
            if (tlb[i].pageNumber == pageNumber) {
                frameNumber = tlb[i].frameNumber;
                tlbHits++;
                tlbHit = true;
                break;
            }
        }

        // If TLB miss, check the page table
        if (!tlbHit) {
            // Check if the page number is present in the page table
            auto pageTableEntry = pageTable.find(pageNumber);
            if (pageTableEntry != pageTable.end()) {
                frameNumber = pageTableEntry->second;
            } else {
                // Page fault, generate a random frame number and update the
page table
```

```
                frameNumber = rand() % PAGE_TABLE_SIZE;
                pageTable[pageNumber] = frameNumber;
                pageFaults++;
            }

            // Update the TLB with the new page number and frame number
            tlb[tlbIndex].pageNumber = pageNumber;
            tlb[tlbIndex].frameNumber = frameNumber;
            tlbIndex = (tlbIndex + 1) % TLB_SIZE;
        }

        // Calculate the physical address and read the value from physical
memory
        unsigned int physicalAddress = (frameNumber * PAGE_SIZE) + offset; //
Changed data type to unsigned int
        int value = readFromPhysicalMemory(physicalAddress);

        // Write the physical address to the file
        physicalAddressesFile << physicalAddress << std::endl;

        // Output the result
        std::cout << "Logical address: " << logicalAddress << ", Physical
address: " << physicalAddress << ", Value: " << value << std::endl;
    }

    // Output statistics
    std::cout << "TLB Hits: " << tlbHits << std::endl;
    std::cout << "Page Faults: " << pageFaults << std::endl;
    std::cout << "Page Fault Rate: " << static_cast<double>(pageFaults) /
totalAddresses << std::endl;

    return 0;
}
```

OUTPUT



```
logical_addresses.txt
36812
47596
22826
59988
401
38950
57152
33775
62852
63908
```

Randomly generated logical addresses which is stored in a file named **"logical_addresses.txt"**.

Output statistics:

After processing all the logical addresses, the code outputs the following statistics:

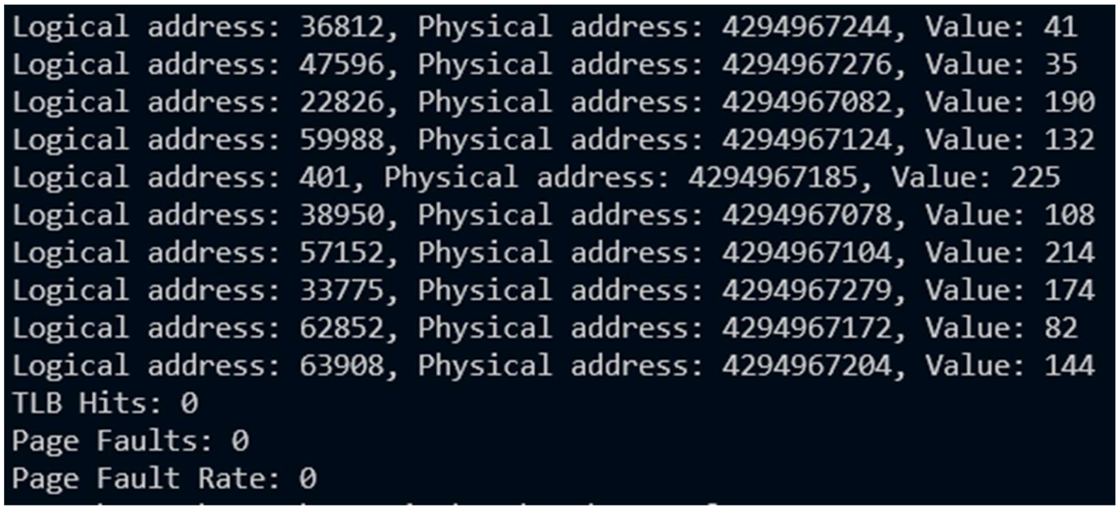
TLB Hits: The number of TLB hits encountered during address translation.

Page Faults: The number of page faults that occurred.

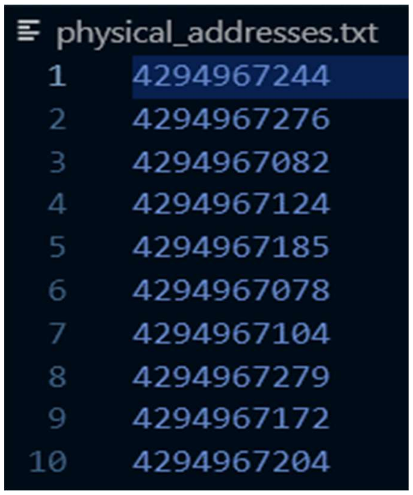
Page Fault Rate: The page fault rate, calculated as the ratio of page faults to the total number of addresses processed.

Output based on above logical address dataset:

The output will vary based on the randomly generated addresses and the behavior of the memory management system. Here's an example of what the output might look like:



```
Logical address: 36812, Physical address: 4294967244, Value: 41
Logical address: 47596, Physical address: 4294967276, Value: 35
Logical address: 22826, Physical address: 4294967082, Value: 190
Logical address: 59988, Physical address: 4294967124, Value: 132
Logical address: 401, Physical address: 4294967185, Value: 225
Logical address: 38950, Physical address: 4294967078, Value: 108
Logical address: 57152, Physical address: 4294967104, Value: 214
Logical address: 33775, Physical address: 4294967279, Value: 174
Logical address: 62852, Physical address: 4294967172, Value: 82
Logical address: 63908, Physical address: 4294967204, Value: 144
TLB Hits: 0
Page Faults: 0
Page Fault Rate: 0
```



```
physical_addresses.txt
4294967244
4294967276
4294967082
4294967124
4294967185
4294967078
4294967104
4294967279
4294967172
4294967204
```

Physical Address Stored in file name called **"physical_addresses.txt".**

Explanation of the output and Time Complexity of the Program

Explanation of the output generated by the code:

Logical address translation and value retrieval:

For each logical address read from the file, the code performs the following steps:

Extract the page number and offset from the logical address.

Check if the page number is present in the TLB. If it is, it is considered a TLB hit, and the corresponding frame number is retrieved.

If the page number is not present in the TLB (TLB miss), the code checks the page table.

If the page number is found in the page table, the corresponding frame number is retrieved.

If the page number is not found in the page table (page fault), a random frame number is generated, and the page table is updated.

The TLB is then updated with the new page number and frame number.

The physical address is calculated by multiplying the frame number by the page size and adding the offset.

The code simulates reading the value from physical memory based on the physical address.

The physical address is written to the "physical_addresses.txt" file.

The logical address, physical address, and value are printed to the console.

Output statistics:

After processing all the logical addresses, the code outputs the following statistics:

TLB Hits: The number of TLB hits encountered during address translation.

Page Faults: The number of page faults that occurred.

Page Fault Rate: The page fault rate, calculated as the ratio of page faults to the total number of addresses processed.

Time Complexity: O (N), where **N** is the number of logical addresses processed.

Conclusion

In conclusion, the task of designing a virtual memory manager involves implementing a program that simulates the translation of logical addresses to physical addresses. The program reads logical addresses from a file, utilizes a Translation Lookaside Buffer (TLB) and a page table to perform the address translation, and retrieves the value stored at the corresponding physical address.

The goal of this project is to gain an understanding of the steps involved in virtual memory management, including address translation, TLB caching, and page faults. By simulating these processes, the program helps demonstrate how a virtual memory system works and how it maps logical addresses to physical addresses.

Overall, this project provides a practical exercise in implementing a virtual memory manager and serves as a valuable learning experience in the field of computer systems and memory management.

References

1. William Stallings. "Operating Systems — Internals and Design Principles", 9th Edition, Pearson publications
2. Andrew S. Tanenbaum, "Modern Operating Systems", Fourth Edition, Pearson publications.
3. Harvey M. Deitel, Paul J. Denel, David R. Choffnes Systems", Third Edition.
4. ChatGPT
5. Bard AI