Designing a Virtual Memory Manager with demand

paging support

# Vecha Devisri Prasad Valluri Veera Gangadhar Sai Teja

***Department of Artificial Intelligence Department of Artificial Intelligence***

***Amrita School Of Engineering Amrita School Of Engineering***

**Bengaluru,Karnataka Bengaluru, Karnataka**

***Abstract*—**This project aims to create a virtual memory manager specifically designed for a multi-user web server environment. It brings together important operating system principles like process scheduling, concurrency control, demand paging, virtual memory, and page tables. The virtual memory manager dynamically allocates memory resources to multiple user sessions, using demand paging techniques to load pages into physical memory only when needed.

Some key features include process scheduling algorithms to ensure fair resource allocation, concurrency control mechanisms to handle simultaneous user interactions, demand paging strategies to optimize memory usage by fetching pages as required, and the implementation of virtual memory and page tables for efficient memory address translation. By integrating these functionalities, the virtual memory manager aims to improve the performance and scalability of the web server, providing seamless user experiences while efficiently utilizing system resources.

*Index Terms*—Virtual Memory Management, Multi-User Systems, Operating System Concepts, Process Scheduling, Concurrency Control, Demand Paging, Memory Allocation, Page Tables,

EAT

## Introduction

In this paper, we’ll take a close look at how virtual memory management works and explore the intricate details of key operating system concepts like process scheduling, concurrency control, demand paging, and memory allocation. Our main goal is to optimize resource usage so that every user gets their fair share of computing power based on their specific needs. We’ll dig deep into the complexities of page tables, uncover the inner workings of demand paging, and embrace the principles of EAT to ensure operational efficiency.

## LITERATURE SURVEY

1. A Survey of Memory Management Techniques in Multiprogramming Systems (1974) by Gerald J. Popek and Robert P. Goldberg

-In this paper, the authors comprehensively review various memory management techniques, focusing on their suitability

for multiprogramming environments where multiple processes compete for system resources. They classify memory management techniques into four categories: relocation, protection, sharing, and logical organization. Through a systematic analysis, they evaluate each technique based on criteria such as efficiency, flexibility, and overhead. The paper provides valuable insights into the trade-offs involved in choosing memory management strategies, considering factors like hardware support, address space allocation, and process isolation.

1. Operating System Concepts: Essentials, Inc. (2018) by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne
   * The authors cover essential aspects of operating systems, including process management, memory management, file systems, and networking. They present these topics with clarity, supplemented by real-world examples and case studies that help reinforce understanding.
2. Computer Systems: A Programmer's Perspective (2009) by Randal E. Bryant and David Warren Arnold
   * The authors start by covering basic concepts such as data representation and assembly language programming before gradually progressing to more advanced topics like memory hierarchy, virtual memory, and system-level I/O. Throughout the book, they emphasize the importance of understanding hardware-software interactions in writing efficient and reliable programs.
3. The VM/370 System by IBM Corporation
   * This system provided users with the ability to create and manage virtual machines, each of which operated as an independent environment with its own operating system instance. VM/370 offered unparalleled flexibility, enabling organizations to maximize the utilization of their mainframe resources while maintaining strong isolation and security between different users and applications.

## Working of the Virtual Memory Description

Virtual Memory Management: This involves the management of memory that appears to be larger than the actual physical memory installed on a system. It allows programs to use more memory than is physically available by temporarily transferring data to disk storage.

Multi-User Systems: These are computer systems that allow multiple users to access and use the system concurrently. Each user typically has their own user account and can perform tasks independently.

Operating System Concepts: These are fundamental principles and mechanisms that underpin the design and functionality of operating systems. They include process management, memory management, file systems, device management, and security.

Process Scheduling: This refers to the method by which the operating system decides which processes to execute when and for how long. It involves algorithms for determining the order and priority of process execution on the CPU.

Concurrency Control: This ensures that multiple processes or threads can execute simultaneously without interfering with each other or causing unexpected behavior. It involves mechanisms such as locks, semaphores, and transactions to coordinate access to shared resources.

Demand Paging: This is a memory management scheme where pages of data are loaded into memory only when they are demanded by the running program. It helps to minimize the amount of memory required to run programs and reduces the time taken to start programs.

Memory Allocation: This involves the management of memory resources in a computer system, including allocating memory to processes, deallocating memory when it is no longer needed, and preventing memory leaks and fragmentation.

Page Tables: These are data structures used by the operating system to map virtual memory addresses to physical memory addresses. They provide the necessary information for the CPU to translate virtual addresses used by processes into physical addresses in memory.

## About the IDE Used to Represent VM

NetBeans IDE is an integrated development environment that provides comprehensive tools for software development, particularly in Java. It offers features such as code editing, debugging, profiling, and project management, all within a userfriendly interface. For your project, NetBeans IDE can serve as a robust platform for writing, testing, and debugging code related to virtual memory management, multi-user systems, and other operating system concepts. Its rich set of plugins and extensions further enhances its capabilities,

making it a valuable tool for developing complex software projects like yours.

## Simulation

### Simulation involves creating virtual environments to mimic real-world systems, while analysis interprets data from these simulations to draw insights into system performance and behavior. These techniques are essential for understanding and optimizing complex systems, such as the multi-user system you’re developing. As part of this, the Producer-Consumer problem was incorporated to simulate thread synchronization, where one thread generates data and another consumes it using a shared buffer. This addition demonstrates inter-process communication and resource coordination, enriching the system’s ability to model real-world concurrency challenges. A. Equations

Effective Access Time (EAT) is a metric used to evaluate the overall efficiency of memory access in a computer system. It combines the time taken to access memory directly (m), the time taken to search the Translation Lookaside Buffer (TLB) for a page translation (E), and the penalty for a TLB miss (E - m \* alpha), where alpha represents the hit ratio.

Here’s a breakdown of each component:

m (Memory Access Time): This is the time required to access data directly from the main memory. E (TLB Search Time): The time taken to search the TLB for a page translation. The TLB is a cache that stores recent translations of virtual memory addresses to physical addresses, speeding up memory access. alpha (Hit Ratio): The probability that a memory access results in a TLB hit, meaning the required translation is found in the TLB cache. It ranges from 0 to 1, where 0 indicates no hits (all accesses result in TLB misses) and 1 indicates all accesses are hits (all translations are found in the TLB). When a memory access results in a TLB hit, the effective access time is simply the time to access memory directly plus the TLB search time (EAT = m + E). However, when there’s a TLB miss, the effective access time increases by the penalty for the miss, which is the time to access memory plus the TLB search time minus the time saved by the TLB hit (EAT = 2m + E - m \* alpha).

Overall, the Effective Access Time provides a comprehensive measure of the average time it takes to access memory in a system, considering both direct memory access and the impact of TLB caching.

## Implementation using NETbeans IDE

The provided code implements a virtual memory management system using Java Swing and AWT libraries within the NetBeans IDE. The system comprises classes for managing pages, memory, and a graphical user interface (GUI) for interaction.

The Page class represents a page in memory, with attributes such as ID, loaded status, assigned memory size, and address. The MemoryManager class handles the management of pages within memory, including functions for loading, unloading, and replacing pages, as well as methods for displaying page information and calculating Effective Access Time (EAT) using

the provided formula. Additionally, it includes a method for searching pages in the Translation Lookaside Buffer (TLB).

The VirtualMemory class represents a virtual memory instance, containing an index and a memory manager. The VirtualMemoryInterface class serves as the GUI interface for creating and managing virtual memory instances. It allows users to input physical memory size, add pages, display the page table, search for pages in the TLB, calculate EAT, and perform optimal page replacement.

The GUI is built using Swing components, with text fields for user input and buttons for performing actions. Each action button is associated with an event listener that triggers the corresponding action in the memory manager. Additionally, the GUI provides a method for performing optimal page replacement, though this functionality is not fully implemented in the provided code.

Overall, the NetBeans IDE facilitates the development of this virtual memory management system by providing a userfriendly environment for coding, debugging, and testing Java applications, enabling efficient development and deployment of the system.

## Implementation Steps

Set Up Development Environment: Install NetBeans IDE. Create a new Java project within NetBeans.

Define Page Class: Create a Java class named Page. Define attributes for the page ID, loaded status, assigned memory size, and address. Implement methods for loading, unloading, and accessing page attributes.

Define MemoryManager Class: Create a Java class named MemoryManager. Define attributes for the page table, memory capacity, alpha, m, and E parameters. Implement methods for handling page requests, loading/unloading pages, replacing pages, adding/removing pages, displaying page table, searching TLB, calculating EAT. Implement memory allocation algorithms like demand paging and page replacement policies such as optimal page replacement.

Define VirtualMemory Class: Create a Java class named VirtualMemory. Define attributes for the memory index and memory manager. Implement methods for accessing memory attributes.

Define VirtualMemoryInterface Class: Create a Java class named VirtualMemoryInterface, extending JFrame. Define attributes for the number of virtual memories. Implement a constructor to create the main GUI window. Implement methods for creating virtual memory instances, opening memory managers, and performing actions like adding pages, displaying page tables, searching TLB, calculating EAT, and performing optimal page replacement. Associate event listeners with GUI components to trigger corresponding actions.

Implement Main Method: Create a main method within the VirtualMemoryInterface class or a separate class. Use SwingUtilities.invokeLater() to initialize the GUI interface.

Memory Allocation and Paging Concepts: Implement demand paging in the MemoryManager class, where pages are loaded into physical memory only when needed. Utilize page tables to map logical addresses to physical addresses, ensuring efficient memory access. Implement memory allocation algorithms such as first-fit, best-fit, or worst-fit to allocate memory resources to pages. Integrate paging concepts such as page faults, page replacement, and TLB (Translation Lookaside Buffer) for optimizing memory usage and access times. Consider implementing additional features like memory segmentation or segmentation with paging for more advanced memory management.

Testing and Debugging: Compile and run the application. Test each functionality to ensure proper operation. Debug any errors or issues encountered during testing.

Optimize and Refine: Refactor code for clarity and efficiency. Optimize algorithms for memory management and page replacement. Enhance the user interface for better usability. By following these comprehensive implementation steps, you can develop a robust virtual memory management system that incorporates memory allocation and paging concepts effectively within the NetBeans IDE.

Step-1)

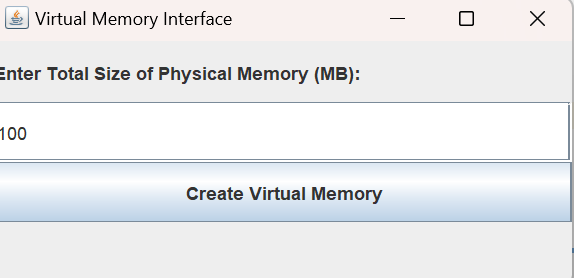


Fig.1) The OS Asks the user to enter the Total size that is to be assigned to the physical memory.

Step-2)

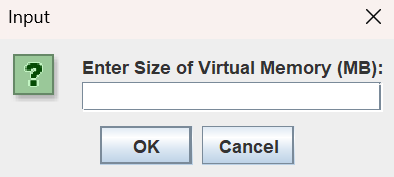


Fig.2) After Entering the Physical Memory, the Virtual Memory Input box where we can assign only limited space i.e. limited to Physical Memory space.

Step-3)

A screenshot of a computer

AI-generated content may be incorrect.

Fig.3) This interface opens instantly after assigning the size of Virtual Memory.

Step-4)

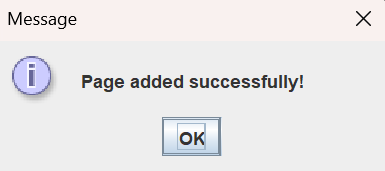


Fig-4) After entering the page id and size and then pressing add page button we get this message.

Step-5)

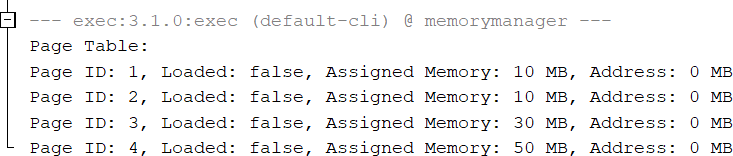


Fig-5) Display Button displays all the pages that are present in the Virtual Memory.

Step-6)

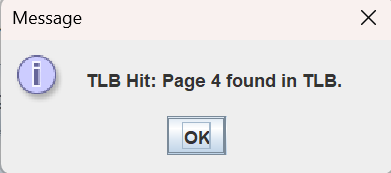


Fig.6) Translation lookaside buffer hit i.e. the page is found and when it is found it is said to be a TLB hit.

Step-7)

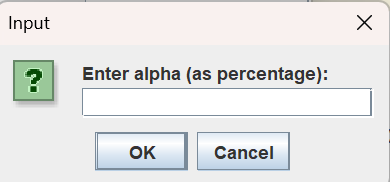


Fig.7) Enter Alpha=Hit Ratio

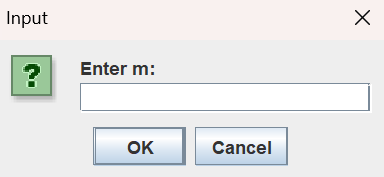


Fig.8) Enter m=Memory Access Time

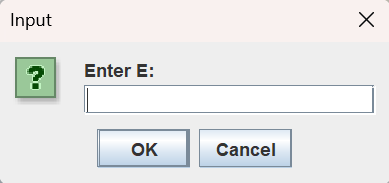
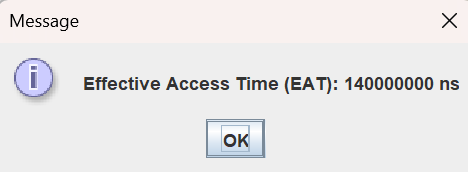


Fig.9) Enter E=TLB Search Time

When the Calculate EAT button is pressed these windows open and as for the values of **Alpha, m and E** Calculates the EAT and then displays the Effective Access Time(EAT) in nanoseconds.



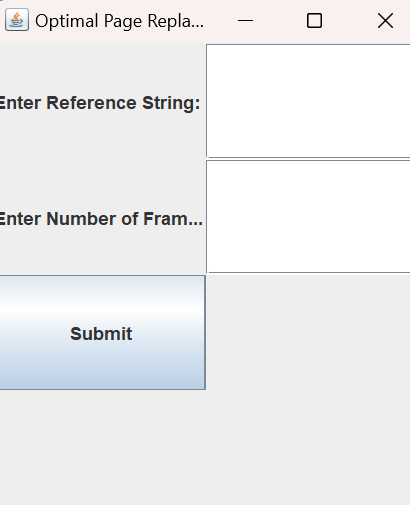
Fig.10) Message stating EAT in nanosecond. Step-8)

Fig.11) The Optimal Page Replacement replaces the page which is not used in the longest dimension of future and displays it according to the given Reference String and with the number of frames.

Step-9)

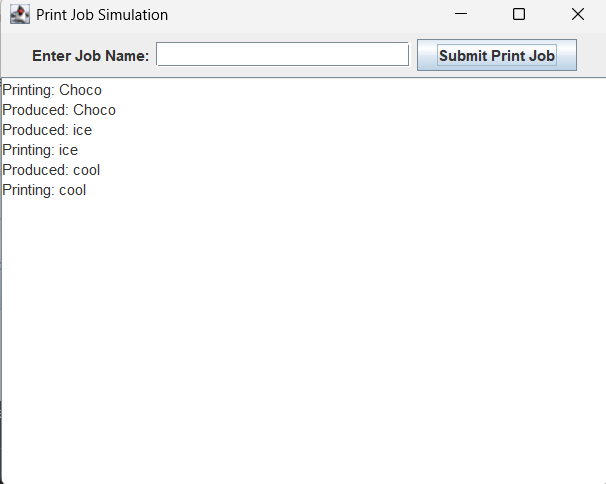


Fig.12) This is the Synchronization technique called Producer-Consumer Problem. If the queue full then producer should wait until the queue is empty. If the queue is empty, then consumer should wait until the queue is full

## Conclusion and Analysis

The implemented code constitutes a Virtual Memory Interface application in Java, providing a graphical user interface (GUI) for managing virtual memory systems. It allows users to create multiple virtual memory instances, each with its own memory manager for managing page operations, such as loading, unloading, and replacing pages.

## Key Features:

**Virtual Memory Creation:** Users can create multiple virtual memory instances by specifying the total size of physical memory and the size of each virtual memory.

**Memory Manager**: Each virtual memory instance is associated with a memory manager, which handles page operations, page table management, TLB (Translation Lookaside Buffer) search,

effective access time calculation, and optimal page

replacement.

**Page Management**: Users can add pages to each virtual memory, display the page table, search for pages in the TLB, and calculate the effective access time for memory operations. Optimal Page Replacement: The application supports optimal page replacement by allowing users to input a reference string and the number of frames for simulation.

## Analysis:

Object-Oriented Design: The code follows object-oriented principles, with classes like Page, MemoryManager, and VirtualMemory encapsulating related functionality. This design promotes code organization, reusability, and maintainability.

Graphical User Interface (GUI): Swing is used to create a simple GUI for user interaction, enhancing the user experience by providing a visual interface for memory management tasks.

Error Handling: Basic error handling is implemented to handle cases such as invalid input for memory sizes and page operations. However, more robust error handling could be added to handle unforeseen scenarios gracefully.

Scalability: The application supports the creation of multiple virtual memory instances, allowing for scalability in managing different memory configurations simultaneously.

Optimization: The implementation of optimal page replacement demonstrates consideration for performance optimization in memory management, crucial for real-world applications handling large datasets.

Overall, the Virtual Memory Interface provides a practical tool for understanding and simulating virtual memory systems, offering users a hands-on experience in memory management concepts through a user-friendly interface. Further enhancements could include additional memory management algorithms, visualization of memory utilization, and improved error handling for a more comprehensive learning experience.

## REFERENCES

1. A Survey of Memory Management Techniques in Multiprogramming Systems (1974) by Gerald J. Popek and Robert P. Goldberg(https://doi.org/10.23860/thesis-armitage- william-1974)
2. Operating System Concepts: Essentials, Inc. (2018) by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne(<https://doi.org/10.14778/3415478.3415571>)
3. Computer Systems: A Programmer's Perspective (2009) by Randal E. Bryant and David Warren Arnold[(https://thuvienso.hoasen.edu.vn/handle/123456789/](https://thuvienso.hoasen.edu.vn/handle/123456789/8621) [8621](https://thuvienso.hoasen.edu.vn/handle/123456789/8621))
4. The VM/370 System Primer-a study of multiplicity and usefulness by L. H. Seawright and R. A. MacKinnon