A
Project Report
On

DYNAMIC MEMORY MANAGEMENT SIMULATOR

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Kamakshi Bhatt	**2261295**
Jighyasa Negi	**2261288**
Nikita Khati	**2261396**
Anjali Joshi	**2261096**

Under the Guidance
of
Mr Prince Kumar
Assistant Professor

Project Team ID: 28

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal,
Uttarakhand March-2025

STUDENT'S DECLARATION

We, **Kamakshi Bhatt, Jighyasa Negi, Anjali Joshi, Nikita Khati** hereby declare the work, which is being presented in the project, entitled '**Dynamic Memory Management Simulator**' in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Prince Kumar.**

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Kamakshi Bhatt	2261295
Jighyasa Negi	2261288
Nikita Khati	2261396
Anjali Joshi	2261096

CERTIFICATE

The project report entitled "Dynamic Memory Management Simulator" being submitted by **Kamakshi Bhatt (2261295), Jighyasa Negi (2261288), Anjali Joshi (2261096), Nikita Khati (2261396)** of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr.Prince Kumar
(Project Guide)

Dr. Ankur Bisht
(Head,CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Mr. Prince Kumar'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Kamakshi Bhatt	2261295
Jighyasa Negi	2261288
Nikita Khati	2261396
Anjali Joshi	2261096

ABSTRACT

The **Dynamic Memory Management Simulator** is an educational software tool designed to visually demonstrate and analyze various dynamic memory allocation strategies used in operating systems. These strategies—**First Fit, Best Fit, Worst Fit**, and **Next Fit**—play a crucial role in determining how memory is allocated and deallocated at runtime, affecting system performance, efficiency, and resource utilization.

This simulator provides an interactive environment where users can allocate and deallocate memory blocks, observe how each strategy behaves under different scenarios, and examine memory fragmentation and compaction in real time. The tool aims to improve conceptual clarity by offering visual feedback and statistics for each operation, making it especially useful for students, educators, and system designers.

Built using modern web technologies, the simulator ensures accessibility, responsiveness, and ease of use. By bridging the gap between theory and practice, it serves as a practical supplement to academic learning in subjects like Operating Systems and Systems Programming.

TABLE OF CONTENTS

LIST OF ABBREVIATION

Abbreviation	Full Form
OS	Operating System
CPU	Central Processing Unit
RAM	Random Access Memory
FIFO	First-In-First-Out
SJF	Shortest Job First
FCFS	First Come First Serve
RR	Round Robin
WT	Waiting Time
TAT	Turn Around Time
GUI	Graphical User Interface
PID	Process ID
I/O	Input/Output
KB	Kilobyte
MB	Megabyte
DMM	Dynamic Memory Management

1. INTRODUCTION

1.1 Prologue

Memory management is a vital component of modern computing systems, especially within operating systems that must allocate and manage memory dynamically among running processes. Efficient memory usage is critical not only for performance but also for resource conservation in systems with limited memory.

The **Dynamic Memory Management Simulator** has been developed as an educational and analytical tool that allows users to interactively explore how dynamic memory allocation and paging mechanisms work. It supports key memory allocation strategies such as **First Fit, Best Fit, Worst Fit**, **and Next Fit** as well as paging systems with **page replacement algorithms** including **FIFO, Optimal, and Clock**.

This simulator provides a real-time, visual representation of memory allocation and page replacement behaviour, helping learners bridge the gap between theory and practice.

1.2 Background and Motivation

In academic settings, memory management is often taught through theoretical lectures and static examples. Students frequently struggle to understand how memory is allocated and deallocated in real systems, or how page faults occur and are handled by replacement algorithms.

The motivation behind this project was to create an **interactive learning environment** that visually represents how memory is managed in both dynamic allocation and virtual memory systems. By simulating different memory management strategies and page replacement algorithms, users can develop a deeper understanding of:

- How memory blocks are allocated and fragmented.
- How different strategies impact performance and memory utilization.
- How page faults occur in a paging system.
- How replacement policies affect system behaviour.

This project aims to provide a **clear, user-friendly, and educational experience** for students, educators, and enthusiasts in computer science and operating systems**.**

2.1 Problem Statement

Understanding memory management is essential in operating systems, but many students struggle to visualize how it works. Traditional teaching methods rely heavily on static diagrams and theory, which often fail to convey the real-time behavior of memory allocation and paging.

Key challenges include:

- **Lack of Visualization**: Students can't observe dynamic changes like fragmentation or page faults.
- **Abstract Concepts**: Ideas such as internal/external fragmentation and page replacement policies are difficult to grasp without interactive examples.
- **Limited Tools**: Few simulators allow comparison between multiple memory allocation strategies or paging algorithms.
- **Disjointed Learning**: Most tools cover either memory allocation or paging, not both together.

Solution:

This project addresses these issues by developing an interactive simulator that combines:

- **Memory allocation strategies**: First Fit, Best Fit, Worst Fit, Next Fit
- **Paging with replacement algorithms**: FIFO, Optimal, Clock
- **Visual feedback** of memory layout, page tables, and fault handling

The simulator aims to provide a practical, engaging learning experience by bridging the gap between theory and real-world system behaviour.

3.1 Objectives and Research Methodology

The project aims to design and implement an educational software tool that simulates how operating systems manage memory dynamically and virtually. It targets both conceptual understanding and visual learning. The core objectives include:

1. **To implement classical memory allocation strategies:**
    - **First Fit**: Allocates the first available memory block that is large enough.
    - **Best Fit**: Allocates the smallest block that fits the process, minimizing internal fragmentation.
    - **Worst Fit**: Allocates the largest available block to avoid creating small unusable holes.
    - **Next Fit**: Similar to First Fit, but continues the search from where it left off.
2. **To simulate paging and page replacement algorithms:**
    - **Paging**: Simulate page-to-frame mapping using a visual page table.
    - **Page Replacement Algorithms**:
        - **FIFO (First-In, First-Out)**

        - **Optimal Replacement**
        - **Clock (Second-Chance)**
3. **To provide real-time visualization:**
    - Graphically display memory blocks, allocation status, fragmentation, and page table mappings.
    - Update visual layout dynamically on user interaction (e.g., allocate, deallocate, replace).
4. **To enable interactive user control:**
    - Let users define process size, request allocations, deallocate memory, trigger compaction, and simulate page references.
5. **To support performance tracking:**
    - Display metrics like memory usage percentage, number of page faults, and page fault rate per algorithm.
6. **To ensure accessibility and educational utility:**
    - Develop as a browser-based simulator using modern frontend technologies.
    - Provide a clean, responsive user interface for students, teachers, and OS enthusiasts.

Research Methodology

The development of this simulator followed a structured methodology aligned with typical software engineering practices:

1. Literature Review

- Studied foundational concepts in operating systems including:
    - Dynamic memory allocation
    - Paging and virtual memory
    - Page fault handling and replacement policies
- Reviewed existing educational simulators and identified limitations such as lack of paging support, poor interactivity, or outdated UI.

2. Requirement Gathering

- Defined user requirements focusing on clarity, interactivity, and support for educational use.
- Established core features like allocation strategy switching, real-time page table updates, and algorithm comparison.

3. System Design

- Designed a modular architecture with:
    - **Frontend**: For visualization and user interaction
    - **Backend/Logic Layer**: Handles memory allocation, page mapping, and algorithm execution

- Designed state management logic to simulate memory block lists, page tables, and access sequences.

4. Technology Stack

- **Frontend**:
  - **React.js**: For component-based UI development.
- **Logic Layer**:
  - **JavaScript**: To implement memory and page replacement logic.
- **Tools**:
  - **GitHub**: For version control and collaboration.
  - **Vite**: For fast development server and build process.

5. Implementation

- Developed memory simulation modules with data structures representing blocks and pages.
- Implemented page replacement logic to simulate reference strings and track faults.
- Created a dynamic frontend that visually responds to backend memory changes.

6. Testing and Validation

- **Unit Testing**: Tested each algorithm with small input cases to ensure correctness.
- **Integration Testing**: Ensured frontend and logic modules interact seamlessly.
- **Manual Testing**: Verified UI behavior, edge cases (e.g., full memory, high fragmentation), and error handling.
- **Test Scenarios**: Used sample reference strings and allocation patterns to compare algorithm performance.

7. User Evaluation

- Conducted peer reviews and feedback sessions with students and instructors.
- Made refinements to UI/UX based on feedback for better clarity and engagement.

This structured approach ensured that the project not only achieved its technical goals but also fulfilled its educational purpose effectively.

4.1 Project Organization

The report is structured as follows:

- **Chapter 1:** Introduction – Overview, motivation, objectives, and methodology.
- **Chapter 2:** Literature Review – Existing work on memory management, scheduling, and paging.
- **Chapter 3:** System Design – Architectural overview and algorithm flowcharts.
- **Chapter 4:** Implementation – Development tools, code structure, and algorithm details.
- **Chapter 5:** Results and Analysis – Experimental setup, test cases, performance metrics, and comparisons.
- **Chapter 6:** Conclusion and Future Scope – Summary of findings, limitations, and scope for improvement.

2. PHASES OF SOFTWARE DEVELOPMENT CYCLE

1.2 Hardware Requirements

To develop and run the simulator efficiently, the following hardware specifications are recommended:

Component	Minimum Requirement
Processor	Intel Core i3 or equivalent
RAM	4 GB (8 GB recommended)
Hard Disk	500 MB free space
Display	13” Monitor (1366x768 resolution)
Input Devices	Keyboard and Mouse
Internet	Required for accessing the web-based simulator

2.1 Software Requirements

The software environment includes both development tools and runtime dependencies:

Software	Description / Version
Operating System	Windows 10 / Linux / macOS
Frontend Framework	React JS
Styling Library	Tailwind CSS
Backend Runtime	Node.js (v14 or later)
Package Manager	npm or yarn
Browser	Chrome / Firefox / Edge (Latest)
Code Editor	Visual Studio Code (Recommended)
Version Control	Git

3. FUNCTIONS AND ALORITHMS USED

Dynamic Memory Management Algorithms :

1. First Fit Algorithm

Concept:

Allocates the first memory block that is large enough for the process.

Steps:

- Traverse the list of free memory blocks.
- Find the first block that is **greater than or equal** to the requested size.
- Allocate memory from this block.
- Reduce the size of the free block (if partially used).

2. Best Fit Algorithm

Concept:

Finds the smallest block that is large enough to fulfill the request, minimizing wasted memory.

Steps:

- Traverse all free blocks.
- Select the **smallest suitable** block (minimizes leftover space).
- Allocate and adjust block size.

3. Worst Fit Algorithm

Concept:

Allocates from the largest available block, attempting to leave usable space for future allocations.

Steps:

- Traverse all free blocks.
- Select the **largest suitable** block.
- Allocate and adjust block size.

4. Next Fit Memory Allocation

- A variation of the First Fit algorithm.

- Starts searching for free memory from the last allocated position.
- Wraps around to the beginning if end of memory is reached without finding a fit.
- Reduces allocation time in large memory systems compared to First Fit.
- May cause higher external fragmentation due to uneven memory usage.
- Suitable for systems with repetitive or predictable memory allocation patterns.

CPU Scheduling Algorithms :

CPU scheduling is the method by which an operating system decides the order in which processes are executed on the CPU. Several algorithms are used to manage process scheduling efficiently, each with its advantages and limitations.

- The **First Come First Serve (FCFS)** scheduling algorithm is the simplest. It operates on a queue-based structure where the process that arrives first is executed first, regardless of its execution time. While easy to implement, FCFS can lead to long waiting times, especially if a short job is stuck behind a longer one — a phenomenon known as the "convoy effect."
- The **Shortest Job First (SJF)** algorithm selects the process with the smallest CPU burst time. This approach minimizes the average waiting time and is optimal in terms of efficiency, but it requires prior knowledge of how long each process will take. Additionally, shorter jobs may continuously arrive, causing longer jobs to wait indefinitely, resulting in starvation.
- The **Round Robin (RR)** algorithm is designed for time-sharing systems. Each process is assigned a fixed time quantum and is preempted if it exceeds this time, allowing other processes to execute in a cyclic manner. This ensures fairness and responsiveness but can suffer from performance issues if the time quantum is not chosen carefully — too small a quantum increases context switching overhead, while too large a quantum behaves like FCFS.

Each of these scheduling algorithms has distinct use cases and trade-offs. An effective scheduling strategy often depends on the specific requirements of the system, such as fairness, response time, throughput, and CPU utilization.

Paging Simulation

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. It breaks memory into fixed-size blocks called frames and divides processes into blocks of the same size called pages.

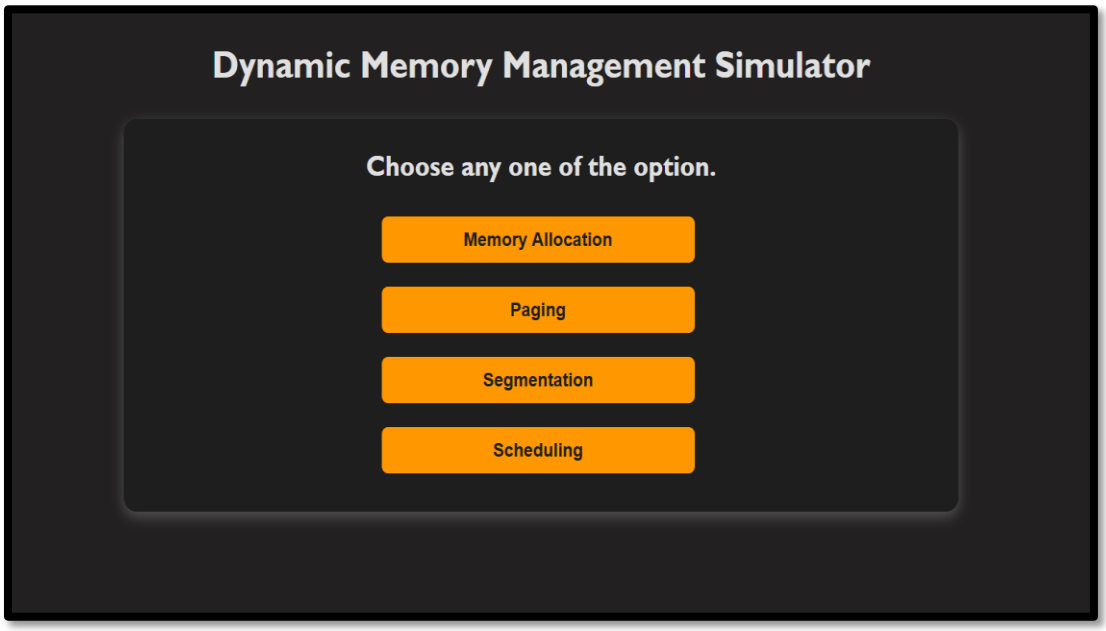
Purpose of simulate Paging :

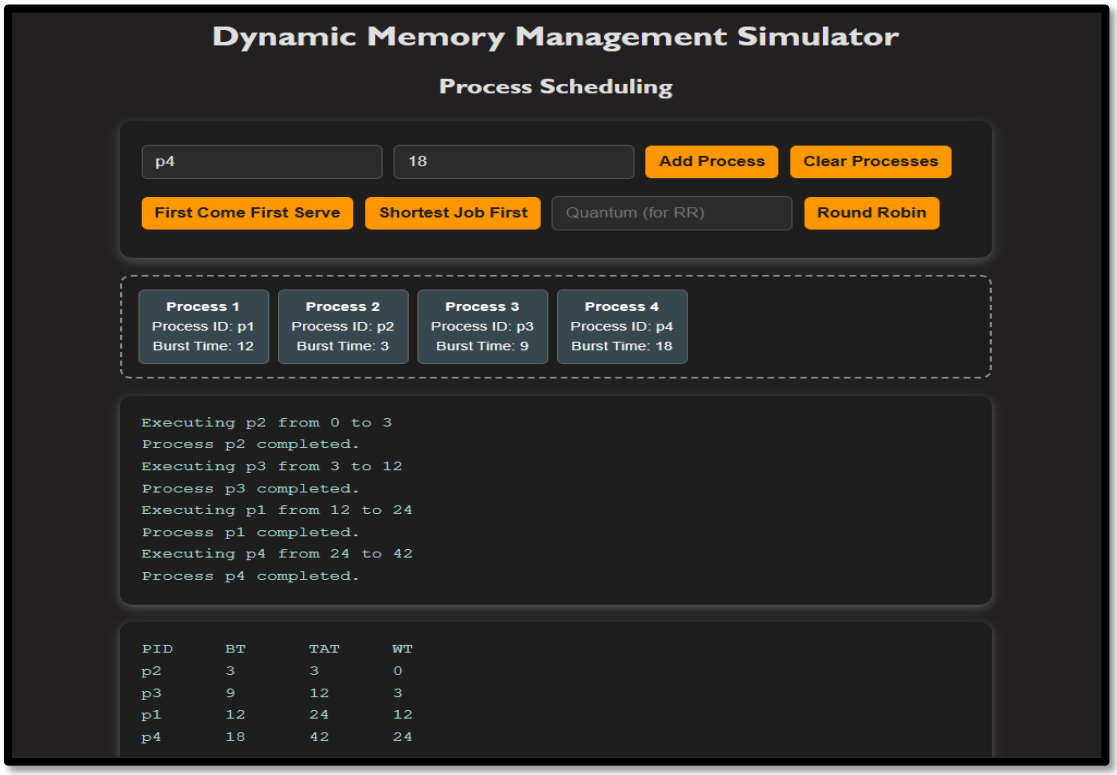
- The simulate Paging function handles the following:
- Breaks down the input string of page references into an array.

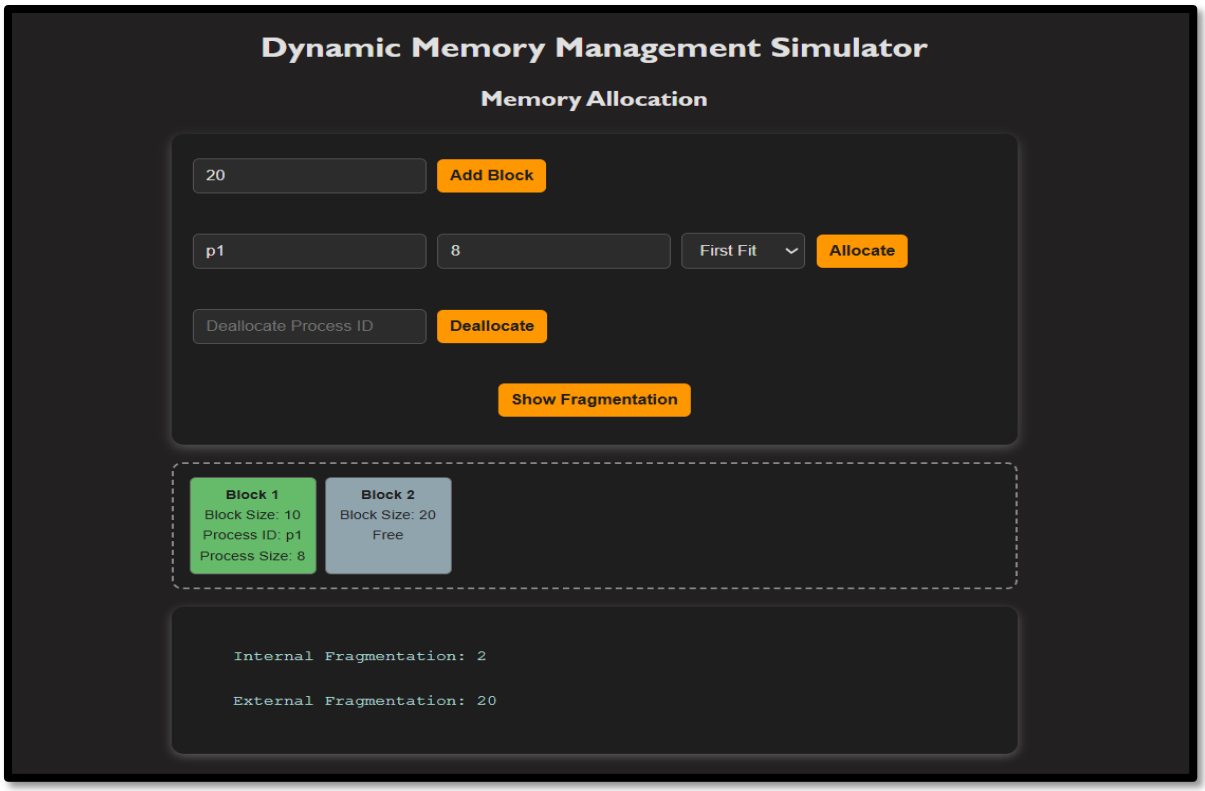
- Simulates memory frame allocation for each page:
- If a page is already in memory, it skips (no page fault).
- If not, it checks for free frames and places the page.
- If memory is full, it removes an old page (e.g., using FIFO/LRU logic if implemented) and loads the new one.
- Updates the UI to visually show how frames are filled over time (e.g., by animating memory blocks).

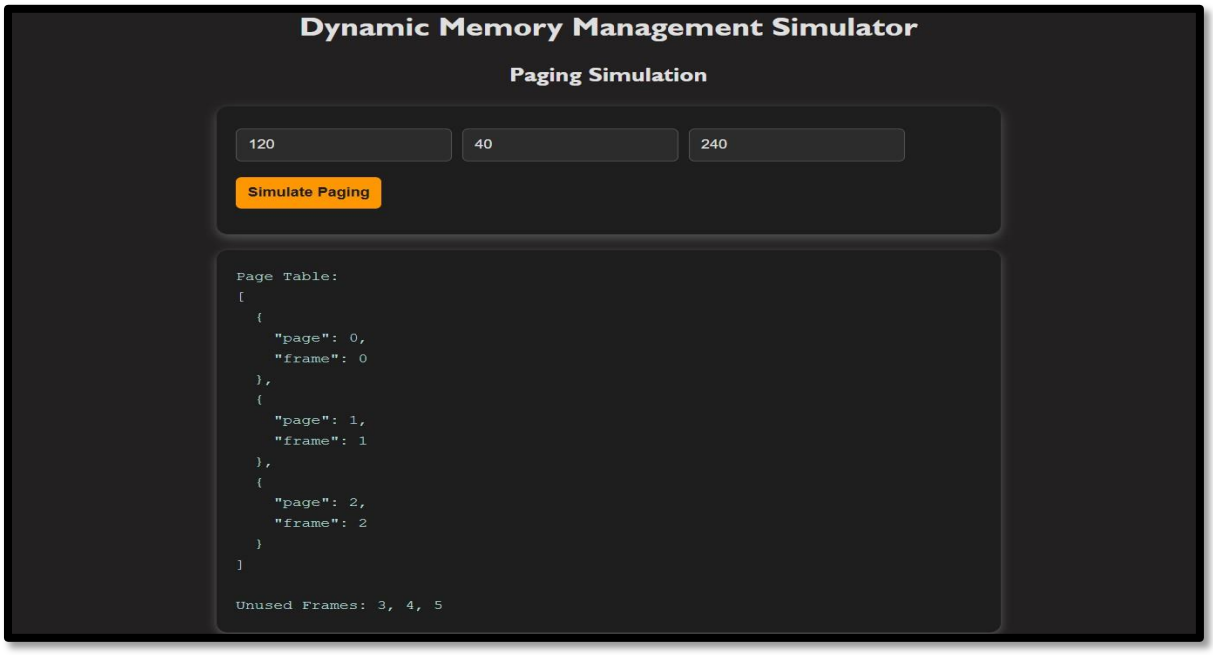
The paging simulation implemented in main.js provides an interactive and educational visualization of how operating systems manage memory using paging techniques. By breaking the user's input into page references and simulating the frame allocation process dynamically, the system bridges theoretical concepts with practical understanding. This component of the simulator emphasizes the importance of non-contiguous memory management and effectively demonstrates how page faults and frame replacements impact system performance. Overall, this simulation serves as a valuable tool for reinforcing core concepts of paging, making abstract ideas more tangible and easier to comprehend.

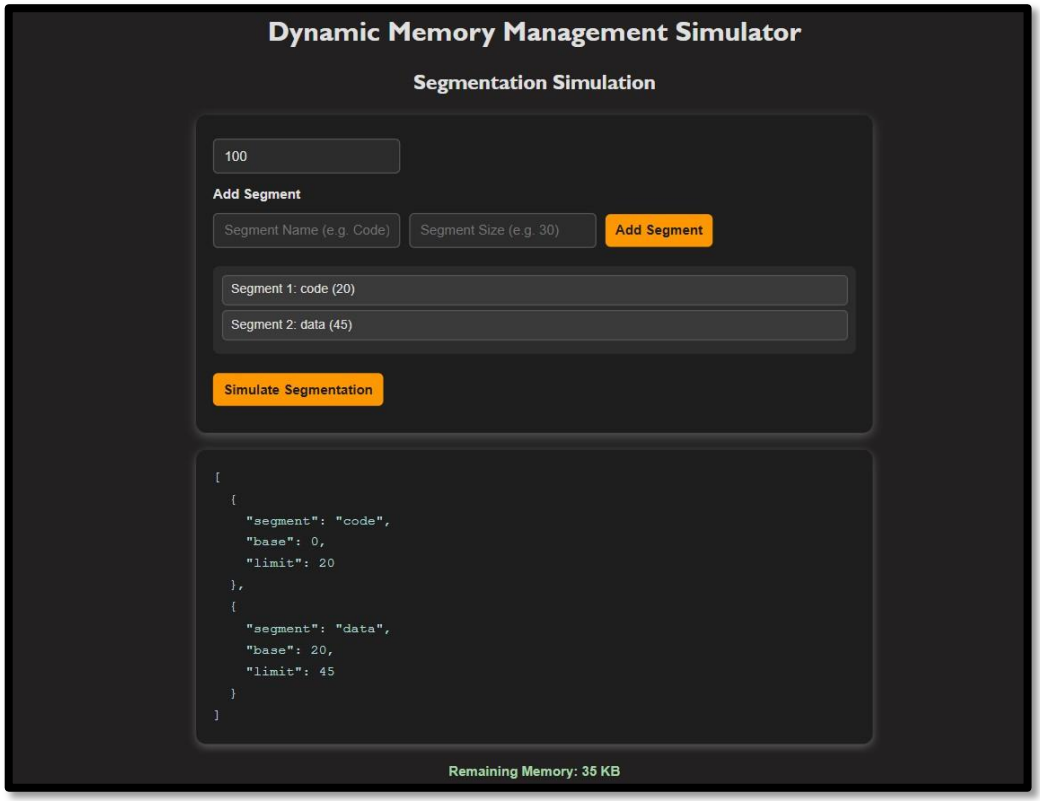
4. SNAPSHOTS


Dynamic Memory Management Simulator
Choose any one of the option.
Memory Allocation
Paging
Segmentation
Scheduling


Dynamic Memory Management Simulator
Process Scheduling
p4
18
Add Process
Clear Processes
First Come First Serve
Shortest Job First
Quantum (for RR)
Round Robin
Process 1
Process ID: p1
Burst Time: 12
Process 2
Process ID: p2
Burst Time: 3
Process 3
Process ID: p3
Burst Time: 9
Process 4
Process ID: p4
Burst Time: 18
Executing p2 from 0 to 3
Process p2 completed.
Executing p3 from 3 to 12
Process p3 completed.
Executing p1 from 12 to 24
Process p1 completed.
Executing p4 from 24 to 42
Process p4 completed.
PID BT TAT WT
p2 3 3 0
p3 9 12 3
p1 12 24 12
p4 18 42 24


Dynamic Memory Management Simulator
Memory Allocation
20
Add Block
p1
8
First Fit
Allocate
Deallocate Process ID
Deallocate
Show Fragmentation
Block 1
Block Size: 10
Process ID: p1
Process Size: 8
Block 2
Block Size: 20
Free
Internal Fragmentation: 2
External Fragmentation: 20


Dynamic Memory Management Simulator
Paging Simulation
120
40
240
Simulate Paging
Page Table:
[
{
"page": 0,
"frame": 0
},
{
"page": 1,
"frame": 1
},
{
"page": 2,
"frame": 2
}
]
Unused Frames: 3, 4, 5


Dynamic Memory Management Simulator
Segmentation Simulation
100
Add Segment
Segment Name (e.g. Code)
Segment Size (e.g. 30)
Add Segment
Segment 1: code (20)
Segment 2: data (45)
Simulate Segmentation
[
{
"segment": "code",
"base": 0,
"limit": 20
},
{
"segment": "data",
"base": 20,
"limit": 45
}
]
Remaining Memory: 35 KB

5. LIMITATIONS

While this project successfully simulates key operating system components, it also has certain limitations that reflect its academic and prototype nature:

1. **Simplified Environment**:
   The project operates in a simulated environment and does not interact with actual hardware or a real operating system. As such, real-world complexities like interrupt handling, I/O operations, and multi-threading are not modeled.
2. **Static Input Data**:
   Process and memory input data are mostly hard-coded or manually entered. The system lacks real-time input generation or process creation during execution, limiting dynamic adaptability.
3. **No GUI Interface**:
   The project is implemented using a command-line interface. While this is sufficient for logic testing, it may not provide the best user experience or visual clarity for memory maps and scheduling charts.
4. **Limited Paging Frame Size**:
   The paging module assumes a fixed number of frames and does not adapt based on system load or memory demand. It also lacks support for multi-level paging or TLB (Translation Lookaside Buffer) simulation.
5. **No Aging in Priority Scheduling**:
   The priority scheduling algorithm does not implement aging, which may result in starvation of low-priority processes over time.
6. **No Real-Time or Multi-core Support**:
   The CPU scheduling is designed for single-core systems. It does not support real-time process deadlines or parallel scheduling across multiple processors.
7. **Basic Memory Allocation**:
   The dynamic memory management focuses only on basic strategies like First Fit, Best Fit, and Worst Fit. Advanced techniques such as Buddy System, memory compaction, or garbage collection are not included.
8. **Performance Not Benchmarked**:
   The efficiency and scalability of algorithms are not tested under heavy or large-scale workloads, which would be necessary for production-level systems.

6. FUTURE ENHANCEMENTS

While the current version of this project effectively simulates core operating system concepts, there is considerable scope for future improvement and expansion to make it more realistic, feature-rich, and aligned with real-world systems. The following enhancements are proposed:

1. **Graphical User Interface (GUI)**:
   Adding a user-friendly GUI using tools like Python Tkinter, JavaFX, or a web-based interface can make interaction more intuitive and allow better visualization of memory allocation, Gantt charts, and page replacement simulations.
2. **Real-Time Process Simulation**:
   Introducing real-time process creation and execution would bring the project closer to real operating systems. This would include process priorities that change dynamically, handling of I/O interrupts, and time-sensitive scheduling.
3. **Support for Multi-core Scheduling**:
   Extending CPU scheduling to support multi-core and multi-threaded environments would significantly increase the project's complexity and realism, allowing parallel processing simulations.
4. **Advanced Memory Management Techniques**:
   Implementation of additional memory allocation strategies such as the **Buddy System**, **slab allocation**, and **garbage collection** would enhance memory management accuracy and system performance.
5. **Paging with TLB and Multilevel Paging**:
   Incorporating Translation Lookaside Buffers (TLB) and multilevel paging would simulate modern memory architectures more accurately. This would also allow tracking of hit/miss ratios and address translation overhead.

7. CONCLUSION

This project successfully demonstrates the fundamental concepts of operating system-level memory and process management through the implementation of dynamic memory allocation strategies, CPU scheduling algorithms, and paging techniques. By simulating these algorithms in a controlled environment, the project offers a practical understanding of how modern operating systems manage limited system resources effectively.

The **dynamic memory management** module provided insights into how different allocation strategies—First Fit, Best Fit, and Worst Fit—affect memory fragmentation and utilization. Through the **CPU scheduling** module, various approaches like FCFS, SJF, Round Robin, and Priority Scheduling were compared based on key performance metrics such as waiting time and turnaround time. Finally, the **paging module** highlighted how page replacement strategies like FIFO, LRU, and Optimal impact system performance, especially under constrained memory conditions.

While the project has certain limitations, such as the absence of real-time processing or a graphical interface, it serves as a strong foundational model for understanding resource management in operating systems. The modular architecture ensures that the system is extensible and can accommodate further enhancements in the future.

Overall, this project bridges theoretical knowledge with practical implementation, providing valuable experience in system-level design, simulation, and performance evaluation. It lays a solid groundwork for future research and development in operating system simulations, memory management strategies, and process scheduling systems.

REFERENCES

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
2. jQuery Foundation. (2020). jQuery API Documentation. jQuery.
3. Mozilla Developer Network. (n.d.). HTML, CSS, and JavaScript documentation. MDN Web Docs.
4. Microsoft. (n.d.). Visual Studio Code documentation. Visual Studio Code.