# MEMORY MANAGEMENT SIMULATION

**Operating Systems Project Report**

## Group- 03 (MMS)

**Report submitted on 04 December 2024**

A project submitted to **Dr. FARAH JAHAN, Professor,** and **SHIMA CHAKRABORTY, Assistant Professor,** Department of Computer Science and Engineering, Chittagong University (CU) in partial fulfillment of the requirements for the Operating Systems Lab course. The project is not submitted to any other organization at the same time.

| Student_Id | Name | Signature | Date |
|---|---|---|---|
| 20701011 | Robin Dey | Robin | 04-12-2024 |
| 20701063 | Ifthekhar Hossain | Ifthekhar | 04-12-2024 |
| 20701072 | Md Mizbah Uddin | Mizbah | 04-12-2024 |
| 21701021 | Md. Moin Uddin | Moin Uddin | 04-12-2024 |
| 21701077 | Mohammad Rakib | Rakib | 04-12-2024 |

Table 1: Details of Group- 03 (MMS)

# Contents

# List of Figures

# List of Tables



# Listings

**Abstract**

**Memory Management Simulation** is a critical component of operating system design, ensuring efficient utilization of memory resources while maintaining system performance and stability. Simulation of memory management techniques provides an insightful way to analyze and compare various algorithms, such as paging, segmentation, and dynamic partitioning, under controlled conditions. This process allows researchers and developers to assess factors like memory allocation, fragmentation, and swapping efficiency without interfering with real-world systems. By replicating complex memory operations, simulation facilitates an understanding of how virtual memory, page replacement policies, and memory protection mechanisms interact to optimize resource usage. Furthermore, it offers a platform for testing new memory management strategies and predicting their impact on different workloads. The insights derived from these simulations contribute to the development of more robust and efficient operating systems, addressing challenges like scalability, real-time processing, and multi-user environments.

# 1 Introduction

We have developed the **"Memory Management Simulation"** for a deeper understanding of memory management techniques in operating systems. Memory management is a critical aspect of modern computing, enabling the efficient allocation, utilization, and management of a computer's memory resources. This simulation provides a platform for analyzing the behavior of various memory management algorithms, enhancing our understanding of their principles, challenges, and performance in real-world scenarios. The objective of this project is to create a practical tool for visualizing and experimenting with these techniques, applying the theories learned in **Computer Architecture** and **Operating Systems** courses.

## 1.1 Background and Motivation

Memory management is at the core of an operating system's functionality, ensuring optimal use of limited memory resources while maintaining system performance and stability. Challenges like fragmentation, deadlocks, and inefficient allocation make it vital to study and simulate memory management algorithms in a controlled environment. By creating a memory management simulation, users can visualize and understand the processes of allocation, deallocation, and memory protection. This project aims to bridge the gap between theoretical concepts and practical implementation, providing insights that contribute to the design of efficient memory systems.

## 1.2 Problem Statement

Efficient memory management is essential for the seamless operation of computing systems, yet it remains a challenging area due to the complexity of modern workloads and the limitations of memory resources. Key issues include memory fragmentation, inefficient allocation policies, and lack of real-time visibility into memory usage. These challenges lead to degraded system performance, increased latency, and underutilization of memory resources. Addressing these issues requires a detailed understanding of memory management techniques, which can be achieved through a simulation-based approach.

- **Proposed Solution:** We propose the development of a Memory Management Simulation tool to address these challenges. The tool will enable users to experiment with and analyze various memory management techniques, such as paging, segmentation, and dynamic partitioning. By simulating different scenarios and workloads, the tool

will provide insights into memory utilization, allocation efficiency, and system stability under different conditions.

- **Objectives:**

  1. Develop an interactive simulation tool for visualizing memory management techniques.
  2. Demonstrate the behavior of allocation and deallocation algorithms.
  3. Highlight challenges like fragmentation and provide strategies to mitigate them.
  4. Enable users to compare the efficiency of different memory management techniques.
  5. Provide a platform for testing new memory management strategies in a controlled environment.

## 1.3 System Definition

The Memory Management Simulation system is a software tool designed to mimic the functionality of an operating system's memory manager. It serves as an educational platform for understanding the intricacies of memory allocation, deallocation, and protection. The system allows users to interact with and observe the impact of various memory management algorithms under different conditions, providing a hands-on learning experience.

- **Purpose and Objectives:**

  1. **Visualization:** Provide a graphical interface to visualize memory allocation and fragmentation.
  2. **Algorithm Testing:** Allow users to test and compare different memory management algorithms, such as first-fit, best-fit, and worst-fit.
  3. **Performance Analysis:** Measure and report metrics such as memory utilization, allocation efficiency, and fragmentation levels.
  4. **Scenario Simulation:** Simulate various scenarios, including real-time and batch processing, to observe algorithm behavior.

## 1.4 System Development Process

The development process of the Memory Management Simulation tool follows a structured approach to ensure functionality, usability, and accuracy.

1. **Requirements Gathering:** Identify key functionalities, such as memory allocation, deallocation, and fragmentation visualization, by engaging with stakeholders and analyzing system requirements.
2. **System Design:** Define the architecture, user interface, and data flow. Design algorithms for memory management techniques and decide on the simulation platform (e.g., desktop or web-based).
3. **Prototype Development:** Develop a prototype to validate the design and gather feedback for improvements.
4. **Development:** Implement the backend logic for memory algorithms, create a user-friendly interface, and ensure system scalability.
5. **Testing:** Conduct unit, integration, and system testing to ensure accurate simulation and reliable performance under various scenarios.
6. **Deployment:** Release the tool for educational and research purposes, accompanied by documentation for users and developers.

## 1.5 Organization

The organization of the Memory Management Simulation system encompasses its architecture, user roles, and functional modules.

1. **System Architecture:** The system follows a client-server model where the client interface communicates with the backend logic that implements memory management algorithms.
2. **User Roles:** Users include educators, students, and researchers. Role-based access is implemented for interacting with different system functionalities.
3. **Functional Modules:**
    - Memory Allocation Module: Implements algorithms like first-fit, best-fit, and paging.
    - Visualization Module: Provides a graphical representation of memory allocation, deallocation, and fragmentation.

- Reporting Module: Generates reports on memory utilization and algorithm performance.
- Scenario Module: Allows users to define custom scenarios for testing and analysis.

4. **Security and Authentication:** Role-based access and secure authentication mechanisms ensure data integrity and prevent unauthorized usage.

5. **Alerts and Notifications:** The system provides alerts for issues like high fragmentation or low available memory, guiding users to take corrective actions.

# 2 Project Management

Organization and management of resources:
We used GitHub to manage the resources regarding this project. Resources are shared among the members using the following link: GitHub repository. The repository has 3 files, "OS_Project_final_1.c", "OS_Project_final_1.pdf" and "OS_Group_03_lab_report" for sharing resources among members. Additionally, Trello is used to assign work to every individual and track their progress.

**Roles of Each Member:**

- **Ifthekhar Hossain:** Implementing the feature of memory allocation and deallocation and helping other team members to implement the rest of the features.
- **Robin Dey:** Implementing the feature of virtual memory allocation and helping other team members to implement the rest of the features.
- **Md Mizbah Uddin:** Implementing the feature of the memory map and the memory statistics and helping other team members to implement the rest of the features.
- **Md. Moin Uddin:** Implementing the feature of internal and external fragmentation and helping other team members to implement the rest of the features.
- **Mohammad Rakib:** Implementing the feature of the performance matrix and helping other team members to implement the rest of the features.

The team worked on combining each feature in a single code that can help show the overall features of our project in the command prompt. Regular meetings were held to discuss progress, troubleshoot challenges, and ensure alignment with project goals.

# 3 Implementation

This section describes the implementation of the Memory Management Simulation project. The project includes various memory management techniques such as dynamic memory allocation, paging, segmentation, virtual memory, fragmentation visualization, and performance metrics.

## 3.1 Dynamic Memory Allocation (Week 1)

**Objective:** To simulate how memory is allocated and freed at runtime.

**Implementation:** Dynamic memory allocation was implemented using the standard C functions `malloc()`, `calloc()`, `realloc()`, and `free()`. These functions allow for the allocation and deallocation of memory blocks during runtime. The core logic involved tracking memory blocks allocated and freed and displaying the status of memory blocks on the user interface (UI).

**Team Member Responsibilities:**

- **Ifthekhar:** Implemented the core logic for dynamic memory allocation, including the `malloc()` and `free()` functions.
- **Mizbah:** Developed the GUI for visualizing the allocated memory blocks in real time.

## 3.2 Paging and Segmentation (Week 2)

**Objective:** To simulate memory division techniques.

**Implementation:** Paging divides memory into fixed-size blocks called pages, and segmentation divides memory into variable-sized segments. For paging, we implemented a simple page table mechanism where virtual addresses are mapped to physical addresses. For segmentation, we tracked segments and their respective base and limit addresses. The GUI displayed virtual and physical memory mapping, highlighting the relationship between memory addresses.

**Team Member Responsibilities:**

- **Robin:** Implemented the paging mechanism, managing the page tables and handling page faults.
- **Moin Uddin:** Developed the segmentation mechanism and designed the user interface for memory division visualization.
- **Rakib:** Integrated both paging and segmentation into a cohesive visualization system.

## 3.3 Virtual Memory Simulation (Week 3)

**Objective:** To simulate how pages are swapped between RAM and disk.

**Implementation:** Virtual memory was simulated by swapping pages in and out of disk storage based on page faults. We used a simple page replacement algorithm (e.g., FIFO or LRU) to manage which pages should be moved between RAM and disk. A file system simulation was created to act as the disk. The GUI showed when pages were loaded into RAM and when they were swapped out to disk.

**Team Member Responsibilities:**

- **Ifthekhar:** Implemented the core logic for page swapping, including the page replacement algorithm.
- **Mizbah:** Designed the graphical representation of virtual memory and page swapping between RAM and disk.

## 3.4 Fragmentation Visualization (Week 4)

**Objective:** To visualize internal and external fragmentation.

**Implementation:** Fragmentation occurs when free memory blocks are split into small, non-contiguous spaces. We tracked both internal fragmentation (unused space within allocated memory blocks) and external fragmentation (unused memory outside allocated blocks). A graphical representation was developed to show fragmented memory areas.

**Team Member Responsibilities:**

- **Robin:** Developed the logic for detecting and calculating internal and external fragmentation.
- **Moin Uddin:** Created the visualization of fragmented memory on the GUI, indicating areas with unused or fragmented space.

## 3.5 Performance Metrics (Week 4)

**Objective:** To track memory utilization and page fault rates.

**Implementation:** The performance of the memory management system was measured by tracking memory utilization (the percentage of memory in use) and the rate of page faults (how often the system had to swap pages in and out of virtual memory). We calculated and displayed these metrics in real time on the GUI for users to monitor.

**Team Member Responsibilities:**

- **Rakib:** Implemented the performance tracking logic, including memory utilization and page fault tracking.
- **Ifthekhar:** Integrated performance metrics into the GUI for real-time display of these statistics.

## 3.6 Integration and Final Project (Week 4)

**Objective:** To integrate all features and present the full project.

**Implementation:** In the final week, all features (dynamic memory allocation, paging, segmentation, virtual memory simulation, fragmentation, and performance metrics) were integrated into one cohesive simulation. Debugging and testing were performed to ensure that all functionalities worked seamlessly. The project was then presented as a fully integrated system with a unified user interface.

**Team Member Responsibilities:**

- **All Team Members:** Integrated all features, tested the system, and worked on the final presentation.

# 4 Problem Statement

Efficient memory management is a critical component of modern operating systems. It involves allocating, managing, and deallocating memory resources effectively to ensure optimal performance and resource utilization. However, understanding and implementing memory management techniques can be challenging due to the abstract nature of these processes and their intricate algorithms.

Operating systems use various memory management strategies, such as paging, segmentation, and dynamic partitioning, to handle memory allocation. These strategies must address challenges such as fragmentation, deadlock, and thrashing, which can degrade system performance.

The lack of hands-on tools to visualize these concepts often hinders students and developers from gaining a deep understanding of memory management. Without a practical means of simulating memory allocation and usage, it becomes difficult to:

1. Visualize memory allocation techniques (e.g., first fit, best fit, worst fit).
2. Understand the impact of fragmentation on system performance.
3. Analyze the trade-offs between different memory management strategies.

To bridge this gap, a memory management simulation tool is essential to provide users with an interactive environment for experimenting with and analyzing memory allocation techniques and their effects on system behavior. This tool will serve as an educational aid for students and a prototyping environment for developers working with memory management algorithms.

# 5 Methodology

## 5.1 Requirements Analysis

This phase establishes the foundation for the simulation tool by defining its purpose, scope, and features.

1. **Educational Goals:**
    - Understand how memory allocation techniques work.
    - Analyze the impact of fragmentation on system performance.
    - Explore trade-offs between different algorithms.
2. **Functional Requirements:**
    - Implement core allocation algorithms (First Fit, Best Fit, Worst Fit).
    - Visualize memory allocation, fragmentation, and deallocation in real-time.
    - Allow users to input custom memory sizes and allocation requests.
3. **Non-Functional Requirements:**
    - Ensure the tool is intuitive, fast, and visually appealing.
    - Support extensibility for future algorithms or features.

## 5.2 System Design

The design stage focuses on translating requirements into a blueprint for implementation.

1. **Architecture:**
    - **User Interface (UI):** A graphical interface for user interaction.
    - **Memory Management Module:** Core logic for handling allocation and deallocation requests.
    - **Reporting Module:** Generates insights such as memory usage statistics and fragmentation analysis.
2. **Algorithm Design:**

- **First Fit:** Scans the memory sequentially to find the first block that can accommodate the requested size.
- **Best Fit:** Searches the entire memory to find the smallest suitable block, minimizing unused space.
- **Worst Fit:** Searches for the largest available block, leaving room for larger future requests.

3. **Data Representation:**

   - Arrays for memory blocks if fixed sizes are used.
   - Linked lists for flexible and dynamic memory representation.
   - Metadata for each block, including size, start address, and status (free or allocated).

   **Flow Diagram:** Flowcharts define how the tool handles requests, detects fragmentation, and reallocates memory.

## 5.3 Development

This stage involves writing and testing code for the defined algorithms and components.

- **Programming Language:**
  - **C programming language:** In C, implementing a memory management simulation involves using dynamic memory allocation functions from the 'stdlib.h.h' library. Key functions include 'malloc(size_t size)' for allocating a single block of memory, 'calloc(size_t num, size_t size)' for allocating and initializing memory blocks for arrays,'realloc(void *ptr, sizesize)' for resizing allocated memory, and 'free(void *ptr)' for releasing allocated memory. Data structures like arrays or linked lists can represent memory blocks, where each block tracks attributes such as size, address, and allocation status. Additionally, functions like 'memset' (from 'string.h') can initialize memory, while error handling using 'if' statements ensures proper allocation and avoids memory leaks. These functions form the foundation for implementing algorithms such as First Fit, Best Fit, and Worst Fit.
- **Core Implementation:**

- Write reusable functions for memory allocation and deallocation.
- Create a memory block tracker to monitor current allocations.
- Develop methods to calculate fragmentation dynamically.

**Testing During Development:** Use mock data to verify the correct functioning of algorithms and their interactions.

## 5.4 Visualization

Visualization enhances the usability and effectiveness of the simulation tool.

1. **Graphical Representation:**
   - Display memory as a series of blocks with colors or patterns indicating allocated and free sections.
   - Highlight fragmentation visually to distinguish between internal and external fragmentation.
2. **Interactive Features:**
   - Allow users to input custom requests (e.g., allocate 100KB, deallocate block 2).
   - Enable pausing and resuming to analyze the state of memory after each operation.
3. **Real-Time Updates:**
   - Show the sequence of steps during memory allocation and deallocation.
   - Dynamically adjust the memory display based on operations performed.

## 5.5 Testing

Testing ensures the tool functions as intended across various scenarios.

1. **Unit Testing:**
   - Verify the correctness of allocation and deallocation algorithms.
   - Test the visualization module independently.

2. **Integration Testing:** Ensure seamless interaction between modules (algorithms, UI, visualization).

3. **Test Scenarios:**

   - Allocate memory using different algorithms.
   - Deallocate memory in random sequences.
   - Introduce edge cases (e.g., full memory, tiny memory blocks, large requests).

4. **Performance Testing:**

   - Measure the tool's responsiveness with increasing memory size and request frequency.
   - Evaluate memory and CPU usage for efficiency.

## 5.6 Analysis and Reporting

The tool provides insights into memory management performance through detailed reports.

1. **Key Metrics:**

   - Memory utilization percentage.
   - Internal and external fragmentation levels.
   - Average time for allocation and deallocation.

2. **Comparative Analysis:**

   - Allow users to compare different algorithms based on performance metrics.
   - Generate charts and graphs for visual interpretation.

3. **User Logs:** Maintain a log of user actions to analyze behavior or retrace steps.

## 5.7 Deployment and Documentation

Deployment ensures accessibility, and documentation provides guidance.

1. **Deployment:**

- Package the simulation as a standalone application or web-based tool.
- Ensure cross-platform compatibility.

2. **Documentation:**
   - **User Guide:** Instructions for using the simulation, including input formats, actions, and visual elements.
   - **Technical Documentation:** Details about system design, algorithms, and data structures for developers.

## 5.8 User Feedback and Iteration

Feedback helps improve the tool's functionality and usability over time.

1. **Feedback Collection:**
   - Conduct user surveys or interviews to gather feedback.
   - Use analytics to observe patterns in user interaction.
2. **Iteration:**
   - Address user suggestions by adding new features or refining existing ones.
   - Regularly update the tool to ensure compatibility with evolving platforms.

# 6 Result and Analysis of Memory Management Simulation

The memory management simulation successfully demonstrates the functionality of core memory allocation techniques, including First Fit, Best Fit, and Worst Fit. The results showcase how these algorithms allocate and deallocate memory blocks, handle fragmentation, and optimize resource utilization under varying conditions.

## 6.1 Results

1. **Visualization of Allocation**
   The simulation provides a visual representation of memory blocks, clearly distinguishing between allocated and free spaces. For each allocation request, the algorithm's behavior—whether selecting the first suitable block (First Fit), the smallest suitable block (Best Fit), or the largest available block (Worst Fit)—is accurately reflected.

2. **Fragmentation Analysis**

   - **Internal Fragmentation:** Occurs more prominently in First Fit and Worst Fit due to suboptimal block utilization.
   - **External Fragmentation:** Best Fit reduces external fragmentation but often leaves behind very small, unusable blocks.
   - **Fragmentation Metrics:** The simulation calculates the fragmentation levels, enabling comparisons across algorithms.

3. **Algorithm Efficiency**

   - **First Fit:** Demonstrates faster allocation due to its linear search but results in higher fragmentation over time.
   - **Best Fit:** minimizes fragmentation at the cost of increased allocation time due to exhaustive searches.
   - **Worst Fit:** Delays fragmentation but can create large gaps that become inefficient over time.

4. **Resource Utilization**
   The simulation tracks memory utilization percentages, with Best Fit often showing slightly higher efficiency due to its careful block selection.

## 6.2 Analysis

- **Algorithm Trade-offs:** Each algorithm has strengths and weaknesses depending on the memory request patterns. First Fit is suitable for environments where speed is critical, Best Fit for scenarios requiring optimal memory usage, and Worst Fit for systems anticipating large future allocations.
- **Scalability:** The simulation performs well for small and medium memory sizes but shows increasing allocation time for Best Fit and Worst Fit as memory size grows, highlighting the need for efficient search strategies.
- **Impact of Request Patterns:** Random allocation and deallocation patterns emphasize the significance of addressing external fragmentation, especially for long-running systems.
- **Educational insights:** Users gain a clear understanding of how allocation strategies affect memory performance and resource availability, making the tool effective for teaching and experimentation.

Overall, the simulation provides a comprehensive analysis of memory allocation algorithms, serving as a valuable tool for studying and improving memory management strategies.

# 7 Future Work and Maintenance

While the current simulation effectively demonstrates key memory management algorithms, there are several avenues for future improvement and expansion to enhance its capabilities and usability.

## 7.1 Future Work

- **Support for Additional Algorithms:** Future versions of the simulation could include additional memory management algorithms, such as *Buddy System*, *Slab Allocation*, and *Segmented Paging*, allowing users to experiment with and compare a wider range of strategies.
- **Dynamic Memory Visualization:** Enhancing the visual representation to allow for real-time tracking of memory status, including visual markers for memory access, load balancing, and memory leaks, could provide a deeper insight into memory management techniques.
- **Advanced Fragmentation Handling:** Implementing techniques like *compaction* to address fragmentation could be added to the simulation, allowing users to observe the impact of compaction algorithms on memory usage and performance.
- **Multithreading and Concurrency:** Supporting multithreaded memory allocation and deallocation requests would introduce users to the complexities of concurrent memory access, simulating real-world scenarios where multiple processes interact with memory concurrently.
- **Graphical User Interface (GUI) Enhancement:** The interface could be made more interactive with options for users to adjust parameters like block size, number of processes, and memory utilization. This would make the tool more user-friendly and adaptable to different learning and development needs.
- **Integration with Operating System Simulators:** The tool could be integrated with operating system simulators to enable the simulation of memory management in more realistic, complex environments, involving multiple processes with varying memory requirements.

## 7.2 Maintenance

To ensure the simulation remains useful and accurate over time, the following maintenance activities should be undertaken:

- **Bug Fixing:** Regular checks for bugs and issues that may arise from updates in the operating systems or programming languages used. Bug fixing will ensure the simulation runs smoothly and remains a reliable educational tool.
- **Updating Documentation:** As the tool evolves, documentation should be kept up to date, including both user and technical documentation, to reflect new features and improvements.
- **Performance Optimization:** Ongoing performance evaluations and optimizations should be carried out, especially as new algorithms or features are added. Ensuring that the simulation remains efficient even with larger datasets is critical for long-term usability.
- **User feedback incorporation:** Continuously collecting and incorporating user feedback is essential for refining the tool. Surveys and user reviews can help identify new features, common issues, and areas for improvement.
- **Compatibility Updates:** Ensuring that the simulation remains compatible with the latest operating systems and development environments is vital. Regular updates will help prevent issues with newer OS versions or programming languages.

By addressing these future work opportunities and maintenance tasks, the memory management simulation tool can continue to evolve and serve as a valuable resource for students, developers, and researchers interested in memory management techniques.

# 8 Conclusion

The memory management simulation successfully demonstrates the implementation and behavior of core memory allocation algorithms such as First Fit, Best Fit, and Worst Fit. By providing an interactive and visual representation of memory allocation, fragmentation, and deallocation, the simulation offers a comprehensive tool for understanding the complexities of memory management.

The simulation allows users to visualize the trade-offs between different algorithms, providing valuable insights into their strengths and weaknesses. It helps in understanding how memory allocation techniques affect system performance, memory utilization, and fragmentation. Through the use of real-time updates and visualizations, users can experiment with different scenarios and gain hands-on experience in memory management.

Furthermore, the simulation serves as an excellent educational tool for students learning about operating systems, as well as a practical resource for developers to test and analyze memory management strategies. While the current version covers basic algorithms and functionalities, future improvements, such as supporting additional algorithms, enhancing the user interface, and optimizing performance, will further enhance its usability and learning potential.

In conclusion, this memory management simulation provides an essential platform for understanding and experimenting with memory management techniques, offering a practical and engaging way to explore complex concepts in computer science and operating systems.

# 9 Book References

The following books were referenced during the development of the Memory Management Simulation project. These resources provided valuable insights into memory management concepts and algorithms, which helped guide the implementation of the simulation.

# References


[1] Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, 4th Edition, 2014.

> This book provides a comprehensive explanation of memory management techniques, including partitioning, paging, segmentation, and virtual memory. It was particularly useful for understanding the underlying principles of memory management and how they are implemented in modern operating systems.

[2] William Stallings, *Operating Systems: Internals and Design Principles*, Pearson, 9th Edition, 2018.

> Stallings' book covers a wide range of memory management topics, with detailed explanations of both theoretical concepts and practical implementations. This text served as a foundation for understanding the algorithms used in the simulation, such as page replacement algorithms and memory allocation techniques.

[3] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne, *Operating System Concepts*, Wiley, 10th Edition, 2018.

> Silberschatz et al. provide in-depth coverage of memory management, including fragmentation, paging, and virtual memory. This reference was instrumental in the development of the virtual memory simulation and in learning how modern systems handle memory allocation and deallocation.