CSE 323

Operating System Design

Section: 14

Project Report

Submission Date : Friday,26th December, 2025

Faculty **:** SSI (DR. SAFAT SIDDIQUE)

Student Information

Name : MD. Munwar Asef

ID : 2231980-642

Department : ECE

Email : *munwar.asef@northsouth.edu*

Semester : Fall 2025

OS Memory Simulator

1. Introduction

Operating Systems (OS) form the backbone of modern computing systems by managing hardware resources and providing essential services to applications. Among the core responsibilities of an operating system, **memory management** and **page replacement** play a crucial role in ensuring efficient execution of programs, optimal utilization of main memory, and system stability. Concepts such as fixed-size partitioning, allocation algorithms (First Fit, Next Fit, Best Fit, Worst Fit), page replacement algorithms (FIFO, Optimal), and the Buddy System are fundamental topics taught in undergraduate OS courses. However, these concepts are often abstract and mathematically oriented, making them difficult for students to visualize and fully understand through theory alone.

The **OS Memory Simulator** project is designed to bridge this gap between theory and practice by providing a **graphical, interactive, and animated simulation** of core memory management techniques. Instead of relying solely on textbook examples, this simulator allows users—especially teachers and students—to visually observe how memory is divided, how processes are allocated, how fragmentation occurs, how pages are replaced in memory frames, and how the buddy system recursively splits memory blocks using powers of two.

This project integrates three major OS modules into a single unified application:

1. **Fixed Size Memory Partitioning with Allocation Algorithms**
2. **Page Replacement Algorithms (FIFO and Optimal)**
3. **Buddy System Memory Allocation**

2. Requirements Analysis

The requirements analysis phase focuses on identifying what the system must do (functional requirements) and how the system should perform (non-functional requirements). Since this project is an educational simulator, clarity, correctness, and usability are prioritized over low-level performance optimization.

Functional Requirements

1. **Memory Partition Module**
    - Allow the user to define total memory (fixed at 1024 MB).
    - Allow the user to choose the number of fixed-size partitions.
    - Accept partition sizes from the user.
    - Visually divide memory into partitions using proportional graphics.
    - Allow the user to enter process sizes.
    - Implement and animate the following allocation algorithms:
        - First Fit
        - Next Fit
        - Best Fit

        - Worst Fit
    - Display internal fragmentation for each allocated partition.
2. **Page Replacement Module**
    - Allow the user to input a reference string.
    - Allow the user to define the number of frames.
    - Implement FIFO and Optimal page replacement algorithms.
    - Animate each page access step-by-step.
    - Highlight page hits and page faults.
    - Display total page faults at the end.
3. **Buddy System Module**
    - Allow the user to select predefined memory sizes (512 MB, 256 MB, 128 MB, 64 MB).
    - Recursively divide memory blocks into power-of-two sizes.
    - Animate each split visually using graphics.
    - Display a textual summary of the splitting process.
4. **Navigation Requirements**
    - Provide a main menu for selecting modules.
    - Provide back buttons to navigate between menus.

Non-Functional Requirements

- **Usability**: Simple and intuitive GUI suitable for students and teachers.
- **Reliability**: Accurate algorithm implementation according to OS theory.
- **Responsiveness**: Smooth animations without freezing the interface.
- **Portability**: Runs on any system with Python installed.
- **Maintainability**: Modular code structure for future extension.

3. System Design (Diagrams)

The system is designed using a **modular architecture**, where each OS concept is implemented as a separate logical module within a single application. This ensures separation of concerns while allowing smooth integration.

High-Level Architecture

- **User Interface Layer (Tkinter GUI)**
    - Handles user inputs (buttons, text fields)
    - Displays animations and graphics
- **Logic Layer**
    - Memory partition algorithms
    - Page replacement algorithms
    - Buddy system logic
- **Visualization Layer**
    - Canvas-based drawing
    - Color-coded memory blocks

- Step-by-step animations

Flow Design (Textual Diagram)

```
User → Main Menu
        ├── Fixed Partition Module
        │      ├── Partition Input
        │      ├── Process Input
        │      └── Allocation Algorithms
        ├── Page Replacement Module
        │      ├── FIFO Algorithm
        │      └── Optimal Algorithm
        └── Buddy System Module
               └── Recursive Memory Splitting
```

Buddy System Design

The buddy system follows a **recursive binary tree structure**, where each block splits into two equal-sized buddies until the smallest unit is reached. Each split is animated and visually aligned to show parent-child relationships.
(Diagrams such as flowcharts, block diagrams, and recursion trees can be added here.)

4. Implementation (Methodology)

The simulator is implemented using **Python** with the **Tkinter** library for GUI and animation. The development followed an incremental and modular methodology.

Development Steps

1. **GUI Skeleton Creation**
   - Window setup, canvas creation, and menu navigation.
2. **Memory Partition Module**
   - Partitions stored as lists.
   - Allocation algorithms implemented using loops and conditions.
   - Fragmentation calculated as (Partition Size − Process Size).
   - Animated drawing using proportional heights.
3. **Page Replacement Module**
   - FIFO implemented using a queue (deque).
   - Optimal implemented using future reference analysis.
   - Frames updated visually after each reference.
   - Page faults counted dynamically.
4. **Buddy System Module**
   - Recursive function to split memory by powers of two.
   - Canvas animation with delay for each split.
   - Labels added for block identification.
5. **Threading & Animation**
   - Background threads used to avoid GUI freezing.
   - Time delays added for smooth visualization.

The implementation strictly follows textbook definitions, ensuring correctness while emphasizing clarity.

5. Testing and Validation

Testing was carried out to ensure that the simulator works correctly and produces results that match operating system theory.

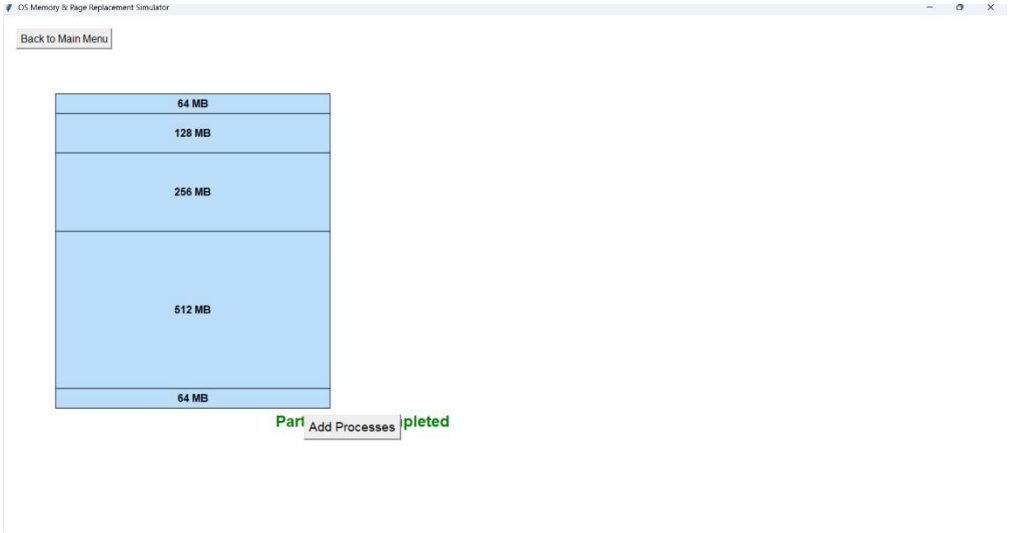
The algorithms were tested using standard examples commonly found in OS textbooks. Different valid and invalid inputs were provided to check whether the system handles errors properly. Visual testing was also important to confirm that memory blocks, partitions, frames, and buddy splits were displayed proportionally and clearly.

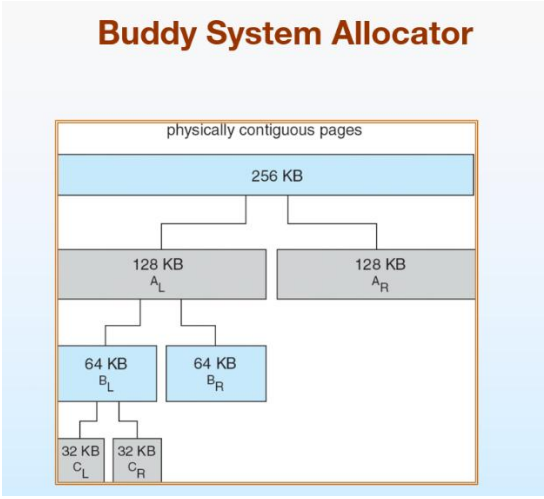
Overall, the simulator behaved as expected and produced correct, stable, and visually accurate results.

6. Results and Screenshots

The simulator successfully demonstrates key operating system memory concepts in a clear and visual manner. Users can observe how memory is partitioned, how processes are allocated, how page faults occur, and how the buddy system divides memory using powers of two.

Relevant screenshots of each module will be added by the author to support these results.)





7. Challenges Faced

Situation

During the development of the OS Memory Simulator, several challenges arose related to animation handling, graphical accuracy, and combining multiple operating system concepts into one application.

Task

The main task was to build an interactive and animated simulator that could clearly demonstrate memory partitioning, page replacement algorithms, and the buddy system without confusing the user or causing the program to become unresponsive.

Action

To address these challenges, careful planning and technical solutions were applied. Animations were designed step by step, calculations were added to scale graphics correctly, and the project was divided into separate modules so that each feature could function independently.

Result

As a result, the simulator runs smoothly, displays accurate visualizations, and allows users to easily switch between different modules without errors or performance issues.

8. How the Challenges Were Solved

Situation

Some simulations involved repeated steps and delays, which risked freezing the graphical interface and reducing usability.

Task

The goal was to keep the application responsive while running animations and calculations in real time.

Action

Threading was used so that animations could run in the background without blocking the main GUI. The code was also modularized, and consistent navigation buttons were added across all

sections. Mathematical scaling was applied to ensure that all memory sizes were displayed proportionally.

Result

These solutions ensured smooth animations, responsive interaction, accurate graphics, and a stable application that effectively demonstrates OS memory concepts.

Conclusion

The **OS Memory Simulator** successfully achieves its objective of providing a comprehensive, interactive, and visually rich platform for learning memory management concepts in operating systems. By integrating fixed partitioning, page replacement algorithms, and the buddy system into a single application, the project offers a holistic view of memory handling techniques.

The simulator enhances conceptual understanding, supports teaching activities, and can be extended in the future to include dynamic partitioning, segmentation, paging, or deadlock simulation. Overall, this project stands as an effective educational tool and a strong demonstration of applied operating system principles.