Software Requirements Specification (SRS)

Project Title

Memory Management Simulator Using Linked List

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document defines the functional and non-functional requirements of the **Memory Management Simulator**. The purpose of this project is to demonstrate how memory is allocated & deallocated in operating systems, and the use of linked list data structure. This document serves as reference for development.

1.2 Scope

The Memory Management Simulator is an educational software designed to model how dynamic memory allocation works in modern operating systems. It allows users to allocate and deallocate memory blocks, demonstrates fragmentation and merging of free blocks, and visualizes memory layout using linked list data structure operations.

1.3 Definitions

Term	Meaning
Memory Block	A segment of simulated memory with size and status
Allocation	Assigning memory to a process
Deallocation	Releasing memory from a process
Free Block	Memory block not in use

Linked List	A dynamic data structure used to represent memory

1.4 References

- IEEE Std 830-1998 – Software Requirements Specification

1.5 Overview of Document

This document is divided into three main sections. The **Overall Description** provides a high-level view of the system. The **Requirements Specification** describes detailed functional requirements. The **Non-Functional Requirements** section defines system quality attributes.

2. Overall Description

2.1 System Environment

The system runs on any standard PC with Python 3 installed. It presents a **console-based interface** for interaction.

2.2 User Classes and Characteristics

User Type	Expertise
Student	Understands basic memory management concepts
Evaluator	Grades correctness and implementation efficiency
Developer	Maintains or enhances the system

2.3 Design Constraints

- Uses Python 3
- terminal UI only
- Linked list used for internal representation

3. Functional Requirements

3.1 Memory Allocation

Description:
The system shall allocate a memory block of requested size to a process.

Requirements:

- The simulator shall prompt the user for process name and size.
- The simulator shall find an appropriate free block using first-fit.
- The allocated block status shall be updated.

3.2 Memory Deallocation

Description:
The system shall free previously allocated memory.

Requirements:

- The simulator shall accept the process identifier to deallocate.
- Adjacent free blocks shall be merged automatically.
- Memory fragmentation must be reduced after deallocation.

3.3 Memory Display

Description:
The system shall display memory layout after operations.

Requirements:

- The display shall show block start address, size, process name, and status (free/allocated).
- Memory should be printed in readable format on the console.

3.4 Error Handling

Description:
The system shall handle invalid input and overflow.

Requirements:

- The simulator shall detect and alert if allocation exceeds available memory.
- The simulator shall validate all input formats and values.

4. Non-Functional Requirements

4.1 Performance

- The system shall process allocations/deallocations in ≤ 1 second.

4.2 Usability

- The simulator shall provide clear menu options and prompts.

4.3 Reliability

- The simulator shall not crash on invalid input; instead display meaningful error messages.

4.4 Maintainability

- The code shall be modular, structured, and documented.

5. External Interface Requirements

5.1 User Interface

- Text-based console menus and prompts.
- No graphical requirements.

5.2 Hardware Interface

- Standard PC hardware.

5.3 Software Interface

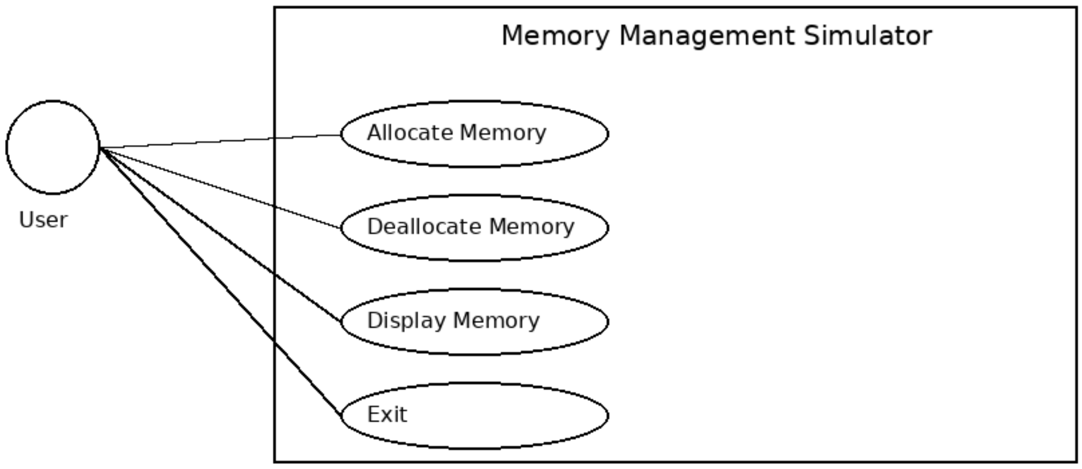
- Python 3 interpreter.

6. Future Enhancements

- Graphical visualization (GUI) using web UI.
- Support different allocation strategies (Best-Fit, Worst-Fit).
- Add memory usage statistics and logging

7. UML Diagrams

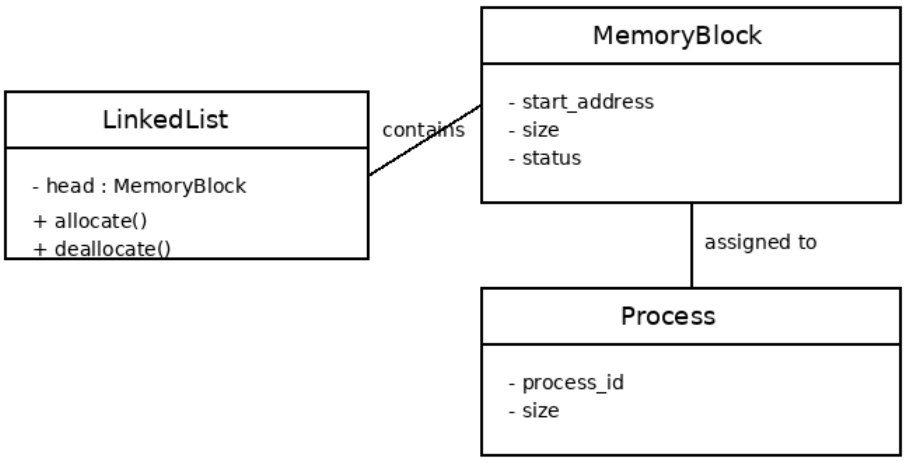
7.1 Use Case Diagram



Brief Description:

This use case diagram illustrates the interaction between the user and the Memory Management Simulator. The user can allocate memory, deallocate memory, view the current memory status, and exit the system through a console-based interface.

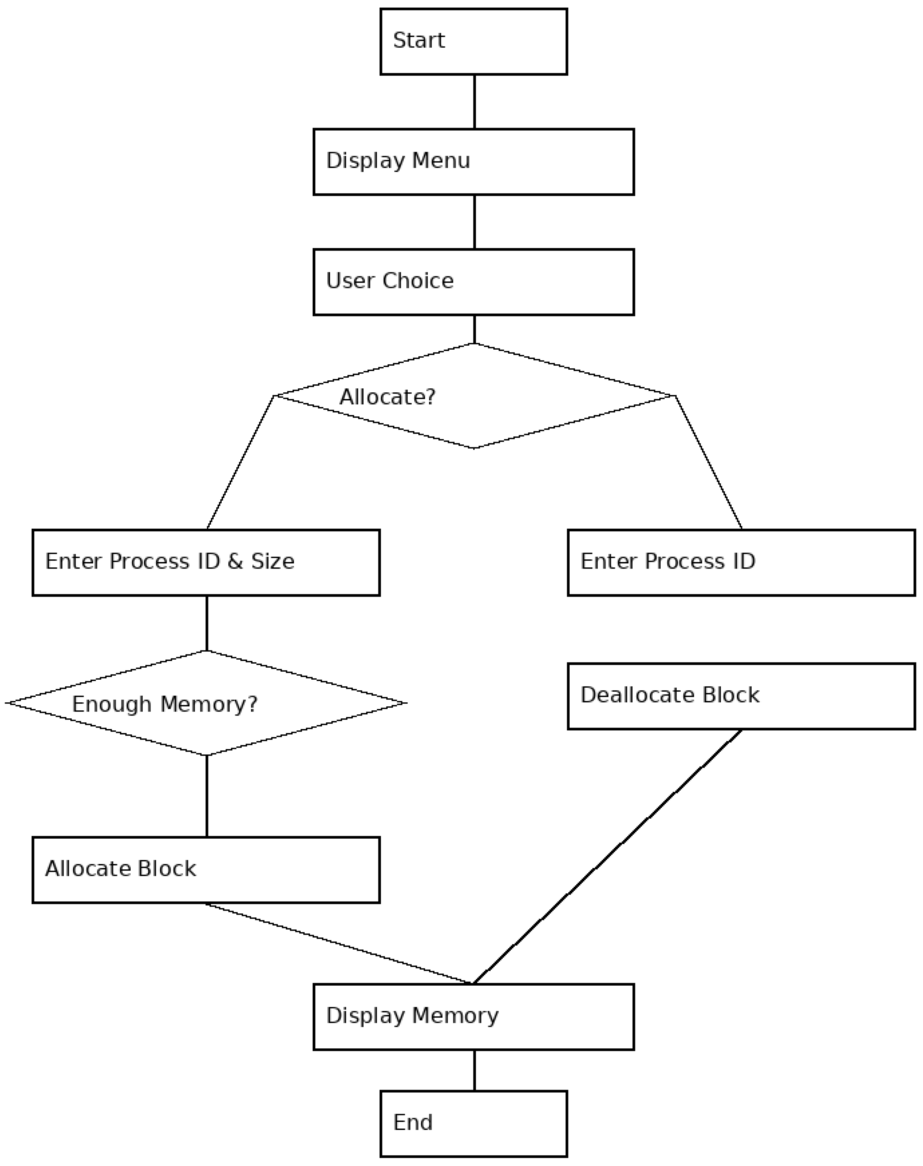
7.2 Object Diagram



Brief Description:

The object diagram depicts the runtime structure of the simulator. It shows how memory blocks are dynamically linked using a linked list and how each allocated memory block is associated with a process during execution.

8. Flow Diagram



Brief Description:
This flowchart illustrates the basic working of the Memory Management Simulator. It shows how the system displays a menu, takes user input, performs memory allocation or deallocation based on the user’s choice, checks memory availability, and finally displays the updated memory status before termination.