Comprehensive Documentation for Memory Management Simulator

This document provides comprehensive documentation for the memory management simulator program written in Python. The program simulates the dynamic allocation and deallocation of memory in a system similar to interpreted languages like Python or Java. It implements techniques like reference counting and memory compaction to manage memory efficiently.

Project Link : Memory_Manager - Shubham Phapale

Assumptions

- The program works with a fixed memory size of 64 MB.
- The input file contains transactions for allocating and deallocating memory blocks.
- Transactions follow a specific format:
- allocate <size>: Allocates a new memory block of the given size in bytes.
- deallocate <variable_name>: Deallocates the memory block associated with the specified variable name.
- print: Prints the current state of allocated and free memory blocks.
- Assignment: Assigns an already allocated block's address to a variable (e.g., a = b).
- The program uses linked lists to manage both allocated and free memory blocks.

Data Structures

- MemoryBlock: A class representing a block of memory, including its starting address, size, and reference count.
- LinkedList: A custom implementation of a linked list to store MemoryBlock objects.

Key Functionalities

- initialize_memory(): Sets up the initial memory state with one free block spanning the entire available memory.
- allocate(size): Attempts to allocate a new memory block of the requested size. It searches for a suitable free block, updates its size and reference count, and adds the allocated block to the appropriate linked list. If no suitable block is found, it attempts compaction before retrying allocation.
- deallocate(variable_name): Finds the allocated block associated with the given variable name, decrements its reference count, and adds it back to the free block list if its reference count reaches zero.
- merge_free_blocks(): Scans the free block list and merges adjacent free blocks to reduce fragmentation.
- compact_memory(): Shifts all allocated blocks to the beginning of memory, consolidating free space at the end. It updates pointers and addresses accordingly.
- process_input_file(filename): Reads and processes transactions from the specified input file, performing actions like allocation, deallocation, and printing memory state.
- print_memory_state(): Prints the current state of both allocated and free memory blocks, including their starting addresses, sizes, and reference counts.

Diagrams :

Diagram 1: Memory Layout Before Allocation

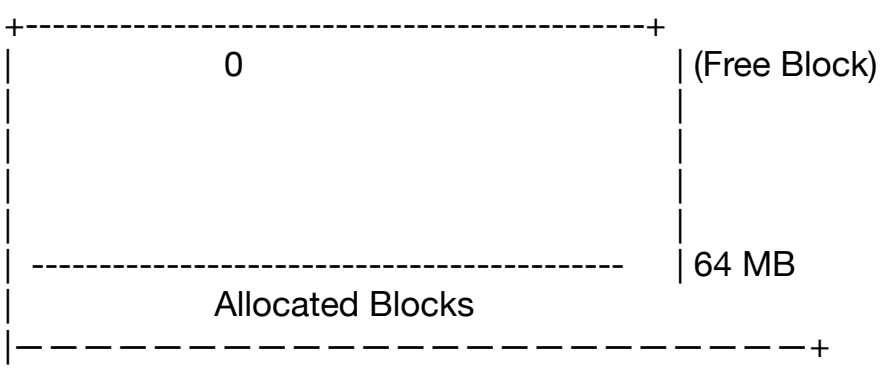


Diagram 2: Memory Layout After Allocation

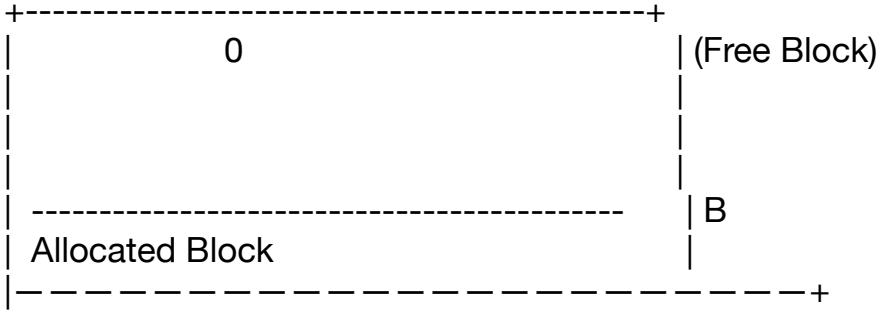
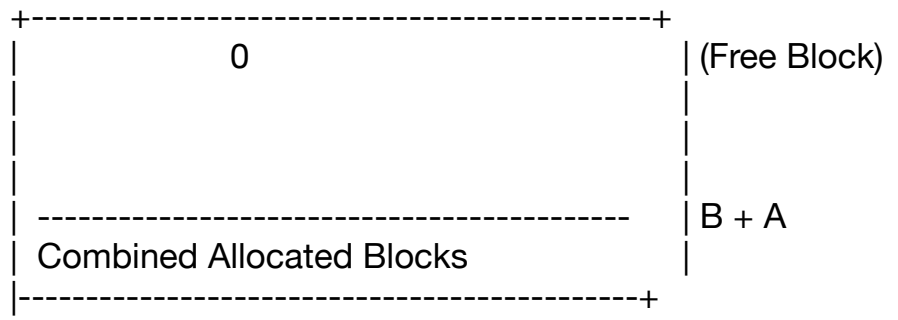


Diagram 3: Memory Layout After Compaction



Code Explanations:

- MemoryBlock class stores crucial information about each memory block, including its address, size, and reference count. Reference count indicates how many variables currently point to the allocated memory.
- Linked lists are used to manage both allocated and free memory blocks efficiently. Insertion, removal, and search operations are straightforward with linked lists.
- allocate function searches for a suitable free block based on size requirements. If found, it updates the block and adds it to the allocated list. Otherwise, it attempts compaction and tries allocation again.
- deallocate function finds the block associated with the given variable name, decrements its reference count, and adds it back to the free list if the reference count reaches zero.
- merge_free_blocks iterates through the free list and combines adjacent blocks, increasing the size of the resulting free block.
- compact_memory shifts all allocated blocks to the beginning of memory, updating pointers and addresses to reflect the new layout. This creates a larger contiguous free space at the end.
- process_input_file reads transaction lines from the file and performs the appropriate actions based on the keyword, such as calling allocate for allocation requests.
- print_memory_state iterates through both allocated and free lists, printing information about each block (address, size, reference_counters).