Design Document: Memory Management Simulator

System Architecture & Implementation Report

January 8, 2026

1 Introduction

This document describes the design and implementation of a comprehensive Memory Management Simulator. The system models the interactions between an Operating System's memory management unit (MMU), a multilevel CPU cache hierarchy, and physical memory allocation. The goal is to simulate OS-level behavior including dynamic memory allocation, cache replacement policies, and virtual memory paging using user-space abstractions.

2 System Architecture

The simulator is modular, consisting of three primary subsystems that interact hierarchically:

1. **Memory Management Unit (MMU):** Handles virtual-to-physical address translation and page fault management.
2. **Cache Controller:** Manages a two-level (L1/L2) cache hierarchy to simulate memory access latency reduction.
3. **Physical Memory Allocator:** Manages the raw physical memory space using dynamic allocation strategies.

2.1 Data Flow

The simulation follows a strictly defined order of operations for memory access:

`User Request(Virtual/Physical Addr)` $\rightarrow$ `MMU` $\rightarrow$ `Physical Addr` $\rightarrow$ `Cache Controller` $\rightarrow$ `Physical Memory`

3 Component Design

3.1 1. Physical Memory Allocation

The physical memory is simulated as a contiguous block of bytes. The simulator implements a **Standard Allocator** using a linked-list data structure to track memory usage.

- **Data Structure:** A singly linked list of `Block` structures. Each block contains metadata: `start_address`, `size`, `is_free`, and a reference to the next block.
- **Allocation Strategies:** The system supports three configurable strategies:
  - **First Fit:** Allocates the first free block that satisfies the request.
  - **Best Fit:** Scans all blocks to find the smallest free block that fits the request, minimizing wasted space.

  - **Worst Fit:** Allocates the largest available free block, leaving large gaps for future use.

- **Deallocation & Coalescing:** When a block is freed (by physical address), the system marks it as free and immediately iterates through the list to merge adjacent free blocks to reduce external fragmentation.

3.2 2. Cache Simulation

The system simulates a configurable multilevel cache (L1 and L2).

- **Configuration:** Each level supports configurable total size, block size, and associativity (Direct-Mapped or Set-Associative). Size of L2 cache is set to 8 times the size of L1 for simplicity.
- **Addressing:** Physical addresses are split into *Tag*, *Index*, and *Offset* bits based on the cache geometry.
- **Replacement Policy:** The cache implements the **Least Recently Used (LRU)** policy. A global access counter is used to track the "age" of cache lines, evicting the line with the smallest timestamp upon a conflict.
- **Write Policy:** The simulation assumes a write-allocate policy where write misses bring the block into the cache.

3.3 3. Virtual Memory (MMU)

The MMU provides a layer of abstraction between the user and physical memory using Paging.

- **Page Table:** Implemented as a `std::map` mapping Virtual Page Numbers (VPN) to Page Table Entries (PTE).
- **Page Table Entry (PTE):** Tracks the Valid bit, Physical Frame Number (PFN), Dirty bit, and Last Access Time.
- **Address Translation:**

$$VPN = \frac{VirtualAddress}{PageSize}, \quad Offset = VirtualAddress \pmod{PageSize}$$

- **Page Fault Handling:** If a VPN is invalid:
  1. The MMU requests a free frame from the Physical Memory Allocator.
  2. If memory is full, the **LRU Page Replacement** algorithm evicts the least recently used page.
  3. If the victim page is "Dirty", a disk write is simulated (symbolic logging).
  4. The new mapping is updated in the Page Table.

4 Implementation Details

The project is implemented in C++ using a modular class structure:

- `MemorySimulator`: Manages the linked list and physical byte array.
- `CacheLevel` & `CacheController`: Manage cache logic and L1/L2 coordination.
- `MMU`: Manages translation and orchestrates interactions between Cache and Memory.
- `main.cpp`: Provides a Command Line Interface (CLI) for user interaction.

5 Assumptions and Limitations

- **Memory Units:** All memory sizes and addresses are represented in Bytes.
- **Page Size:** The Page Size is assumed to be equal to the Frame Size.
- **Disk Backing:** Disk storage is simulated symbolically; evicted pages are effectively destroyed unless re-loaded, and "loading" simply allocates a new zeroed frame.
- **Single Process:** The simulation models a single process environment; context switching is not supported.