Memory Management Simulator

1. Introduction

Modern computer systems rely on several layers of memory management to provide efficient allocation, caching, and address translation. Due to significant differences between physical memory, virtual memory, and cache hierarchies, an educational simulator can help illustrate how these components interact and affect performance.

This project implements a Memory Management Simulator that combines four major subsystems:

1. Dynamic Memory Allocation
2. Buddy Memory Allocation
3. Cache Hierarchy Simulation
4. Virtual Memory with Page Replacement
5. Integrated Virtual Memory + Cache Access

The simulator allows test workloads or interactive commands to explore memory behavior, visualize state changes, and observe statistics such as fragmentation, hit ratios, and page faults.

2. System Overview

The system models memory behavior without hardware-specific details, allowing a simplified but realistic demonstration of how allocation and memory access work. The simulator supports:

1. Commands for allocating and freeing memory
2. Cache read/write operations
3. Virtual address translation
4. Page replacement policies
5. Integrated VM → Cache → Memory flow

Input can be interactive or batch-mode using workload files, making it suitable for lab assignments, testing, and demonstration.

3. Module Descriptions

3.1 Memory Allocator (First Fit / Best Fit / Worst Fit)

This allocator simulates a simple heap-like memory model. Allocation works on a linear address space divided into blocks. Each block contains:

1. Starting address
2. Size
3. Allocation state (free / used)
4. Block ID

Supporting strategies:

- First Fit: choose the first free block large enough
- Best Fit: choose smallest suitable block
- Worst Fit: choose largest suitable block

Free blocks are merged (coalesced) on deallocation to reduce external fragmentation.

The allocator also reports:

- Internal fragmentation
- External fragmentation
- Utilization percentage

This reflects fundamental OS memory management concepts and allows students to compare strategies experimentally.

3.2 Buddy Allocator

The buddy allocator manages memory by splitting blocks into powers-of-two. The goal is to reduce external fragmentation by ensuring that memory can be recombined ("buddied") when free.

Key behaviors:

- Memory is rounded to the nearest power of two
- Allocation size is also rounded up to nearest power of two
- Blocks are recursively split until a suitable size is found
- Freed blocks are merged with their buddy if both are free

Benefits:

- Fast merging
- Reduced fragmentation
- Efficient free-list structure by block size

The simulator reports:

- Internal fragmentation percentage
- Allocation statistics
- Free lists per block size

3.3 Cache Hierarchy

The cache subsystem simulates a multi-level cache with configurable properties:

- Cache size
- Block size
- Associativity
- Replacement policy

Supported replacement policies:

1. FIFO
2. LRU
3. LFU

Cache access attempts to read through levels L1 → L2 → … → Memory. If a miss occurs, data is fetched from memory and inserted into higher levels.

Collected statistics:
Total accesses

- Hits
- Misses
- Hit ratio

The write policy implemented here is a simplified write-through model to keep consistency manageable.

3.4 Virtual Memory System
Virtual memory allows the program to address space larger than physical RAM. The simulator maps virtual pages to physical frames using a software page table.

Supported page replacement algorithms:

- FIFO
- LRU

During address translation:

1. Virtual address → Virtual Page + Offset
2. If page is in memory → hit
3. Otherwise → page fault occurs
4. Either free frame or victim frame is used
5. Page is loaded into memory

Virtual memory statistics include:

- Page hits
- Page faults
- Hit ratio
- Number of loaded pages

3.5 Integrated Virtual Memory + Cache Simulation

A realistic memory access pipeline involves both paging and caching. The simulator combines:

Virtual Address → Virtual Memory → Physical Address → Cache Hierarchy → Main Memory

This integration allows observing effects such as:

- How page faults delay cache access
- How caching improves physical memory access times
- How replacement policies interact

This part mirrors real CPU memory behavior in simplified form.

4. Design and Implementation Details

The simulator is implemented in C++ using object-oriented design. Each subsystem resides in a different module for clarity and testability.

Module Responsibilities:
MemoryAllocator: Linear heap allocation w/ fragmentation tracking
BuddyAllocator: Power-of-two block allocation w/ buddy merging
CacheHierarchy: Multi-level cache with replacement policies
VirtualMemory: Virtual-to-physical translation + page replacement
MemorySimulator: Command interpreter + workflow coordinator

Replacement Policies:
FIFO → Queue
LRU → Timestamp
LFU → Usage counter
CLOCK → Reference bits + hand pointer

Build System:
A GNU Makefile builds the simulator into:

bin/memsim

Workload tests are provided to automate experiments.

5. Testing
The simulator includes several predefined test workloads that automatically exercise the different memory subsystems. These test cases cover allocation strategies, buddy allocation behavior, cache hierarchy access patterns, virtual memory page replacement, and the fully integrated VM + cache pipeline.

Each test workload is structured as a sequence of commands that are fed to the simulator either interactively or in batch mode. The simulator supports redirecting output to .txt files for easier evaluation and comparison.

Video Demo uploaded in github readme