Design and Implementation of a Memory Management Simulator

1. Introduction

This project implements a Memory Management Simulator that models how an operating system manages memory at runtime. The simulator focuses on dynamic memory allocation, cache behavior, and virtual memory using paging, without implementing a real OS kernel.

The goal is to understand:

- Allocation strategies and fragmentation
- Performance trade-offs in memory management
- Interaction between virtual memory, caches, and physical memory

---

2. System Overview

The simulator consists of four major subsystems:

1. Physical Memory Allocator
2. Buddy Memory Allocation System
3. Multilevel Cache Simulator
4. Virtual Memory System

All components are integrated through a command-line interface (CLI) that allows interactive testing.

---

3. Physical Memory Model

3.1 Memory Representation

- Physical memory is simulated as a contiguous byte-addressable array
- Memory addresses are represented as offsets from base address 0
- Memory blocks are managed using explicit metadata structures

Physical Memory: [0 .... N-1]

Each block contains:

- Start address
- Block size
- Allocation status (free/used)
- Block ID

---

4. Dynamic Memory Allocation Strategies

The simulator supports the following allocation strategies:

4.1 First Fit

- Selects the first free block large enough for the request
- Fast but prone to fragmentation

4.2 Best Fit

- Selects the smallest free block that satisfies the request
- Reduces wasted space but increases search time

4.3 Worst Fit

- Selects the largest free block
- Attempts to leave large free regions available

4.4 Allocation Procedure

For each malloc(size):

1. Traverse free block list
2. Select block according to strategy
3. Split block if larger than required
4. Assign block ID and update metadata

4.5 Deallocation Procedure

For each free(block_id):

1. Mark block as free
2. Coalesce with adjacent free blocks
3. Update fragmentation statistics

---

5. Buddy Memory Allocation System

5.1 Design Rationale

The Buddy system is implemented to:

- Eliminate external fragmentation
- Enable fast coalescing using address arithmetic

5.2 Key Properties

- Total memory size is a power of two
- Allocation sizes are rounded up to nearest power of two
- Minimum block size = 8 bytes
- Free blocks are stored in size-indexed free lists

5.3 Buddy Address Calculation

buddy_address = address XOR block_size

5.4 Allocation Flow

1. Round request to power of two
2. Find smallest available block
3. Recursively split larger blocks if needed
4. Allocate and record internal fragmentation

5.5 Freeing and Coalescing

1. Identify buddy using XOR
2. If buddy is free, merge
3. Repeat recursively until merge not possible

5.6 Fragmentation in Buddy System

- Internal fragmentation exists due to rounding
- External fragmentation = 0% (by design)

---

6. Fragmentation and Performance Metrics

The simulator tracks:

6.1 Internal Fragmentation

Internal Fragmentation = Allocated Block Size − Requested Size

6.2 External Fragmentation

External Fragmentation = 1 − (Largest Free Block / Total Free Memory)

6.3 Memory Utilization

Utilization = Used Memory / Total Memory

6.4 Allocation Statistics

- Total allocation requests
- Successful allocations
- Failed allocations
- Allocation success rate

---

7. Multilevel Cache Simulation

Cache	**Size**	**Block Size**	**Latency**
L1	128 B	16 B	1 cycle
L2	512 B	16 B	5 cycles

7.2 Cache Design

- Set-associative cache model
- FIFO replacement policy
- Each access records hit/miss

7.3 Cache Access Flow

CPU → L1 → L2 → Main Memory

- Miss penalties propagate to lower levels
- Hit/miss statistics are tracked per level

---

8. Virtual Memory Simulation

8.1 Address Translation

- Virtual memory implemented using paging
- Page table maps virtual pages to physical frames
- Page size is fixed and documented

8.2 Page Replacement Policy

- FIFO replacement
- Page faults trigger eviction
- Disk access latency simulated symbolically

8.3 Translation Flow

Virtual Address

↓

Page Table Lookup

↓

Physical Address

↓

Cache Access

↓

Memory

8.4 Metrics Tracked

- Page hits
- Page faults
- TLB statistics (optional)

---

9. Command-Line Interface

Supported commands include:

- init memory <size>
- set allocator <first_fit | best_fit | worst_fit | buddy>
- malloc <size>
- free <block_id>
- dump memory
- stats all
- access <virtual_address>

The CLI enables interactive testing and demonstration.

---

10. Assumptions and Limitations

Assumptions

- Single-process simulation
- No concurrency
- Simplified latency model

Limitations

- No real memory access
- No write-back cache modeling
- No multi-process address spaces

---

11. Conclusion

This simulator successfully models:

- OS-level memory allocation strategies
- Fragmentation behavior
- Cache hierarchy performance
- Virtual memory address translation

The project provides hands-on insight into real operating system memory management design decisions and their performance implications.