Memory Management Simulator Design

Ulimella Priyanshu (23124040)

1. Introduction

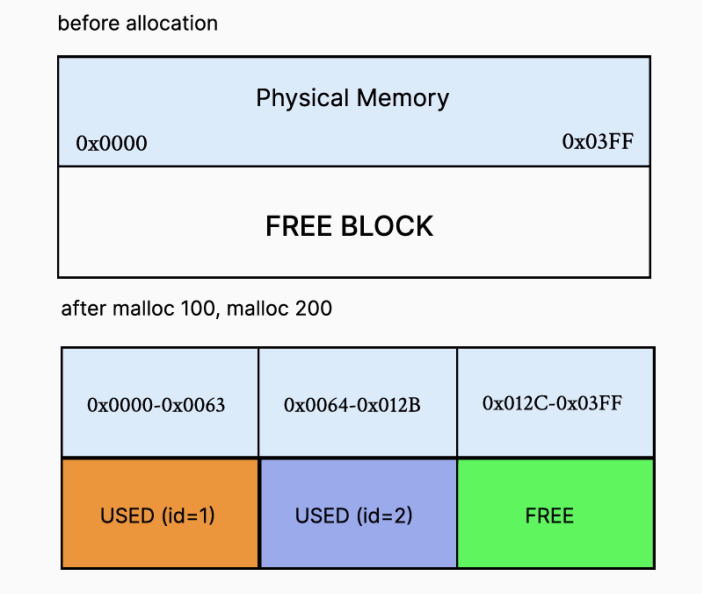
This project implements a user-space simulator that models how an operating system manages physical memory and CPU caches. The simulator focuses on dynamic memory allocation strategies, fragmentation analysis, and multilevel cache behavior. It is not a real OS kernel, but accurately captures OS-level memory management concepts.

2. Memory layout and assumptions

The physical memory is modeled as a contiguous byte-addressable region. Internally, memory is represented using a singly linked list of block descriptors.
Each block stores it's

- Starting address
- Size
- Allocation status
- Block ID

Physical Memory Layout Representation



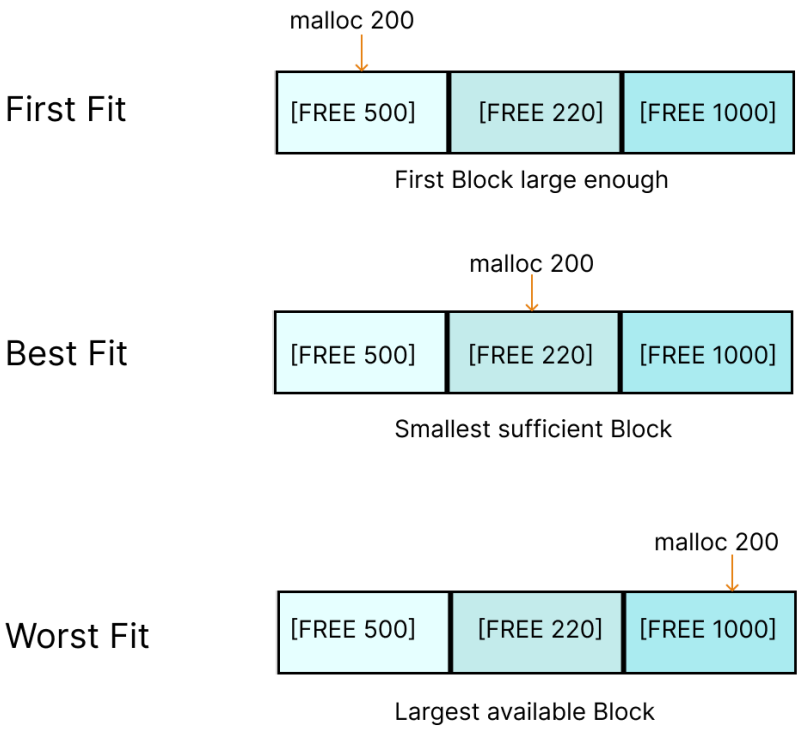
USED = Allocated memory block
FREE = Unallocated memory block
All addresses are byte-addressable.
Allocation and deallocation operations manipulate this block list by splitting or coalescing nodes.

3. Allocation strategy implementations

The simulator supports three dynamic memory allocation strategies

- **First Fit**
  The allocator scans memory from the beginning and selects the first free block that is large enough to satisfy the request.
- **Best Fit**
  The allocator selects the smallest free block that can satisfy the request.
- **Worst Fit**
  The allocator selects the largest available free block.

These strategies determine how free memory blocks are selected when a memory allocation request is made. Memory is represented as a linked list of blocks, where each block stores its start address, size, allocation status, and block identifier.

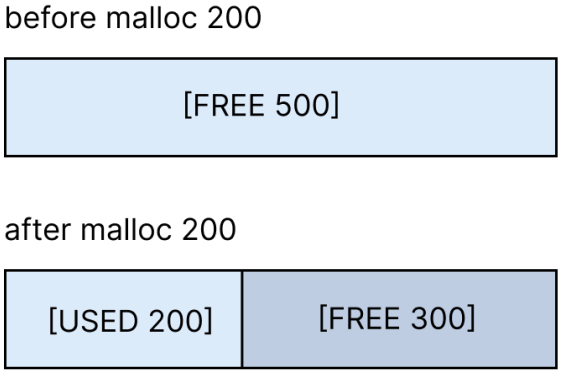


Block Splitting

After selecting a block, if the block size is greater than the requested size, the block is split into:

- an allocated block of requested size
- a remaining free block

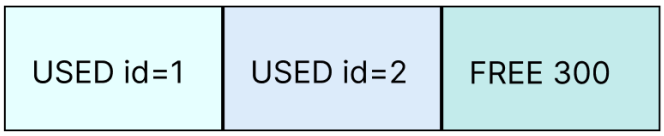
This reduces wasted memory and allows reuse of remaining space.



Coalescing (Merging Free Blocks)

When a memory block is freed, the simulator checks both neighboring blocks.If either neighbor is free, the blocks are merged to form a larger continuous free region.This process minimizes external fragmentation and improves future allocation success.

before free 2



after free 2 (no merge yet)

After Coalescing

4. Cache hierarchy and replacement policy

The simulator implements a simplified **multi-level CPU cache hierarchy** consisting of two levels:

Cache Level	Capacity	Replacement Policy
L1 Cache	Small, Fast	FIFO (First in First Out)
L2 Cache	Large, Slow	FIFO (First in First Out)

Every memory access is first checked in the **L1 cache**. On a miss, the request is forwarded to the **L2 cache**.

This simulator models cache behavior at the memory address level. Each cache line stores a single memory address instead of a block of bytes.

Simplification

To keep the simulator simple and focused on cache behavior. Cache block size, Cache associativity (direct-mapped / set-associative) are abstracted and not explicitly

implemented. Cache blocks represent memory addresses directly; block size and associativity are abstracted for simplicity.

Cache Lookup Flow

The following sequence is used for each `access <address>` command:

1. Check L1 cache
    - If hit → return immediately
    - If miss → forward to L2

2. Check L2 cache
    - If hit → insert address into L1
    - If miss → insert into L2, then insert into L1

FIFO Replacement Policy

Each cache level maintains:

- A list of cached addresses
- A FIFO queue tracking insertion order

When a cache is full:

1. The oldest address (front of queue) is evicted.
2. The new address is inserted.

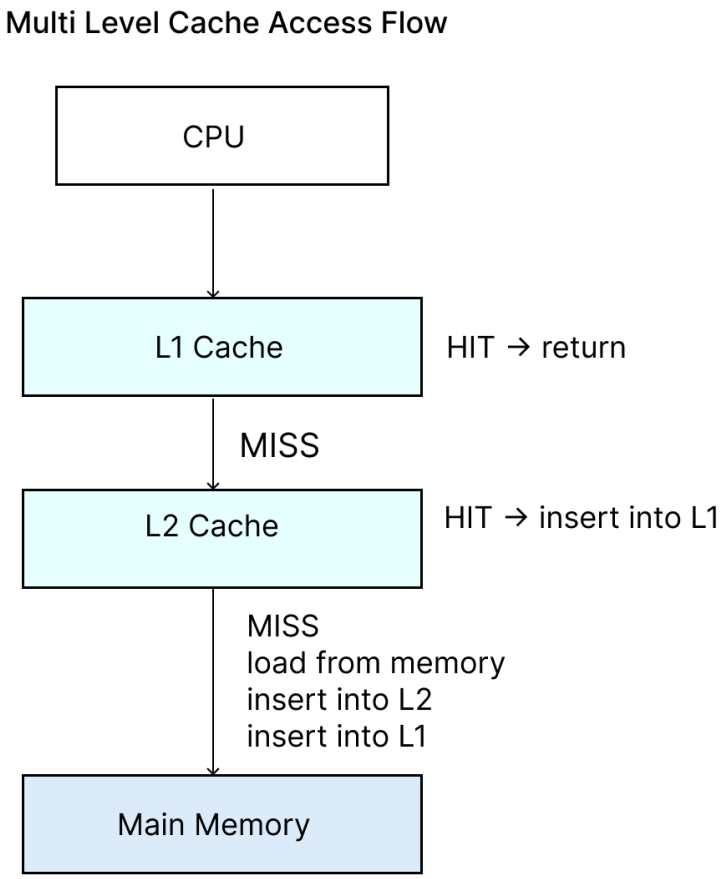
This models a **First-In First-Out (FIFO)** cache eviction strategy.

Statistics Tracked

For every cache level the following are recorded:

- Number of hits
- Number of misses
- Hit ratio

Displayed using the command: cache_stats



5. Translational Flow

The simulator currently accepts physical memory addresses directly and does not implement virtual memory or page tables.In a real operating system, memory access follows the sequence:

Virtual Address → Page Table → Physical Address → Cache → Memory.

This flow is documented here for conceptual completeness.

6. Limitations and simplifications

Area	Limitation / Simplification
Virtual Memory	Virtual memory, page tables, and page replacement algorithms are not implemented. All memory addresses are treated as physical addresses.
Buddy Allocation	The buddy memory allocation scheme is not implemented. Only linked-list based dynamic allocation is supported.
Cache Block Size	Cache blocks are abstracted to represent individual memory addresses instead of fixed-size data blocks.
Cache Associativity	Direct-mapped and set-associative cache behavior is not simulated. All caches operate as fully associative address lists.
Disk I/O & Page Faults	Disk latency and page fault handling are not modeled.
Memory Alignment	Block alignment constraints are ignored.
Concurrency	The simulator is single-threaded and does not simulate concurrent memory access.

Error Handling	Input validation is minimal and assumes well-formed commands.

This simulator uses variable-size block splitting for memory allocation. As a result, allocated blocks always match the requested size exactly, leading to zero internal fragmentation. Internal fragmentation typically arises in fixed-size allocation schemes such as the buddy system, which is listed as a future extension.