Memory Management Simulator: Design & Architecture

-PRATHAM SHARMA(23116076)

1. Memory Layout and Assumptions

The simulator models the computer's **Physical Memory (RAM)** as a linear, contiguous array of bytes. To manage this storage, we implement a **Linked List Memory Map**, where memory is tracked as a sequence of distinct "Blocks."

Block Structure

Each node in our memory list contains:

- **Start Address:** The physical byte index where the block begins.
- **Size:** The total size of the block (User Data + Padding).
- **Padding:** Bytes wasted to satisfy alignment requirements.
- **State:** A boolean flag (is_free) indicating if the block is available or occupied.
- **ID:** A Process ID (PID) to track ownership. -1 denotes a free block.

Key Assumptions

1. **Word Alignment:** All allocations are aligned to a 4-byte boundary. A request for 101 bytes will allocate 104 bytes (101 data + 3 padding).

2. **Initialization:** Memory initializes as a single, giant "Free" block covering the entire requested size (e.g., 4096 bytes).

3. **Address Space:** We simulate a 32-bit physical address space, though the actual container size is configurable at runtime.

2. Allocation Strategy Implementations

The allocator supports three standard algorithms to decide *which* free hole to use for a request.

A. First Fit

- **Algorithm:** Scans the memory list from the beginning. It stops at the **first** hole that is large enough to satisfy the request.
- **Behavior:** Generally the fastest algorithm but tends to accumulate small fragments at the beginning of the memory space.

B. Best Fit

- **Algorithm:** Scans the **entire** list to find the hole that is smallest but still sufficient.
- **Behavior:** Minimizes wasted space in the chosen block but creates tiny, unusable fragments (external fragmentation) and is slower due to the full scan.

C. Worst Fit

- **Algorithm:** Scans the **entire** list to find the **largest** available hole.
- **Behavior:** Ensures that the leftover chunk after splitting is large enough to be useful. However, it quickly destroys large contiguous areas needed for big processes.

Coalescing (Deallocation)

When free(id) is called, the system:

1. Marks the block as free.
2. Checks the **Next** block: If free, merges them.
3. Checks the Previous block: If free, merges them.
4. This is critical to combat external fragmentation.

3. Cache Hierarchy and Replacement Policy

The simulator implements a two-level cache hierarchy (**L1, L2**) backed by RAM.

Structure

- **Set Associative:** Each level is divided into **Sets** (rows) and **Ways** (columns).
- **Inclusive Policy:** Data in L1 is also guaranteed to be in L2 (mostly).
- **Write Allocate:** Write misses fetch the block from RAM and install it in the cache.

Replacement Policy: FIFO (First-In, First-Out)

We do not use LRU (Least Recently Used). Instead, we use a lighter **FIFO** approach:

- Each Set maintains a circular **Victim Pointer** (fifo_next_victim).
- New data is always written to the "Way" pointed to by the victim pointer.
- The pointer then increments: ptr = (ptr + 1) % associativity.
- This ensures that the oldest block in the set is always the one evicted.

4. Address Translation Flow

When the CPU executes a read 0x1A4 command, the simulator translates the address using bitwise operations:

1. Decoding

The 32-bit address is split into three parts:

- **Offset:** log2(Block_Size) bits. Determines the byte inside the block.
- **Index:** log2(Num_Sets) bits. Determines which Row to look in.
- **Tag:** Remaining bits. Acts as the unique ID for the data.

2. Lookup Flow

1. **L1 Check**:

- Go to L1[Index]. Compare Tag against all valid lines.
- Hit: Return immediately (Fastest).
- Miss: Pay latency penalty, go to L2.

2. **L2 Check:**

- Go to L2[Index]. Compare Tag.

- Hit: Copy block to L1. Return.
- Miss: Pay large latency penalty, go to RAM.

3. **RAM Fetch:**

- Retrieve block.
- Install in L2 (evicting old L2 data if needed).
- Install in L1 (evicting old L1 data if needed).

5. Limitations and Simplifications

1. **No Data Storage:** The simulator tracks the *presence* of data (Tags/Valid bits) and *allocation state*, but it does not store actual values (e.g., writing "Hello" to address 0x10 is not supported).

2. **Unified Cache:** We simulate a unified Instruction/Data cache. There is no separation between I-Cache and D-Cache.

3. **No Virtual Memory:** All addresses are treated as **Physical Addresses**. There is no simulation of Page Tables, TLB, or MMU.

4. **Static Latency:** Cache and RAM latencies are fixed constants (1, 10, 100 cycles) and do not account for bus contention or variable DRAM timings.