Design and Implementation of a Memory Management Simulator

Deependra

Contents

1 Overview

This document details the design and implementation of a modular Memory Management Simulator. The system mimics the core responsibilities of an Operating System kernel, including Physical Memory Management (allocation/deallocation), Virtual Memory (paging), and a Multilevel Cache Hierarchy.

The primary goal is to simulate runtime behavior, visualize memory layout, and analyze fragmentation metrics under different allocation strategies (First Fit, Best Fit, Worst Fit).

2 Memory Layout and Assumptions

2.1 Physical Memory Model

The physical memory is abstracted as a contiguous array of bytes, but management is performed using a **Doubly Linked List** structure.

- **Storage:** The raw memory is simulated using a `std::vector<char>` of size N.
- **Tracking:** A linked list of `memoryBlock` nodes tracks allocations.
- **Initialization:** Memory begins as a single "Mega Block" representing the entire free space.

2.2 Data Structures

The core unit of management is the `memoryBlock` structure:

```
struct memoryBlock {
  int free;              // Status flag (1=free, 0=used)
  int block_id;          // Unique identifier for handle
  size_t start;          // Start Address
  size_t sz;             // Block Size
  memoryBlock* prev;     // Pointer to previous block
  memoryBlock* next;     // Pointer to next block
};
```

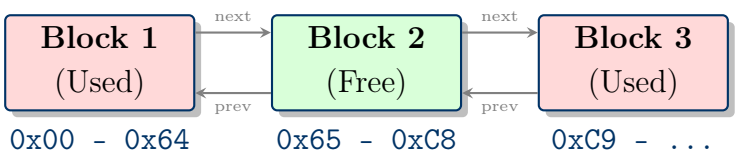

Figure 1: Doubly Linked List Memory Layout

2.3 Assumptions

1. **Addressing:** Addresses range linearly from 0 to $TotalSize - 1$.
2. **Fragmentation:** External fragmentation is calculated based on the ratio of the largest free block to total free memory.
3. **Safety:** Invalid access or double-free errors are caught using an internal ID map (`allocated_blocks`).

3 Allocation Strategy Implementations

The simulator employs the **Strategy Design Pattern**. An abstract base class `allocatorr` defines the interface, allowing runtime switching of algorithms.

Strategies Summary

- **First Fit:** Selects the *first* sufficient block. Fast ($O(1)$ best case) but clusters data at start.
- **Best Fit:** Scans *all* blocks for smallest fit. Saves large gaps but creates tiny fragments.
- **Worst Fit:** Scans *all* blocks for largest fit. Prevents tiny fragments but uses up large spaces.

3.1 Coalescing (Deallocation)

All strategies implement immediate coalescing. Upon freeing a block, the system checks its `prev` and `next` neighbors. If either is free, they are merged into a single contiguous block to reduce fragmentation.

4 Cache Hierarchy and Replacement

4.1 Architecture

The system simulates a two-level cache (L1 and L2) using **Set-Associative Mapping**.

- **L1 Cache:** Small size, small block size (e.g., 32B), low associativity.
- **L2 Cache:** Larger size, larger block size (e.g., 64B), high associativity.

4.2 Addressing

A Physical Address (PA) is decomposed into:

1. **Tag:** Unique identifier for the memory block.
2. **Set Index:** Determines which cache set to check.
3. **Offset:** Position within the cache line.

4.3 Replacement Policy: FIFO

The simulator enforces a **First-In, First-Out (FIFO)** replacement policy.

- Each `cache_line` stores an `inserted_at` timestamp.
- On a cache miss in a full set, the line with the oldest timestamp is evicted.
- Unlike LRU, accessing a line (Cache Hit) does *not* update the timestamp.

5 Virtual Memory Model

5.1 Paging Mechanism

The `virtual_memory_manager` decouples Virtual Addresses (VA) from Physical Addresses (PA).

- **Page Table:** A vector of `page_table_entry` structs.
- **Entry Metadata:**
  - `frame`: The mapped physical frame number.
  - `valid`: Boolean flag (Present/Absent).
  - `loaded_at`: Timestamp for replacement.

5.2 Address Translation Flow

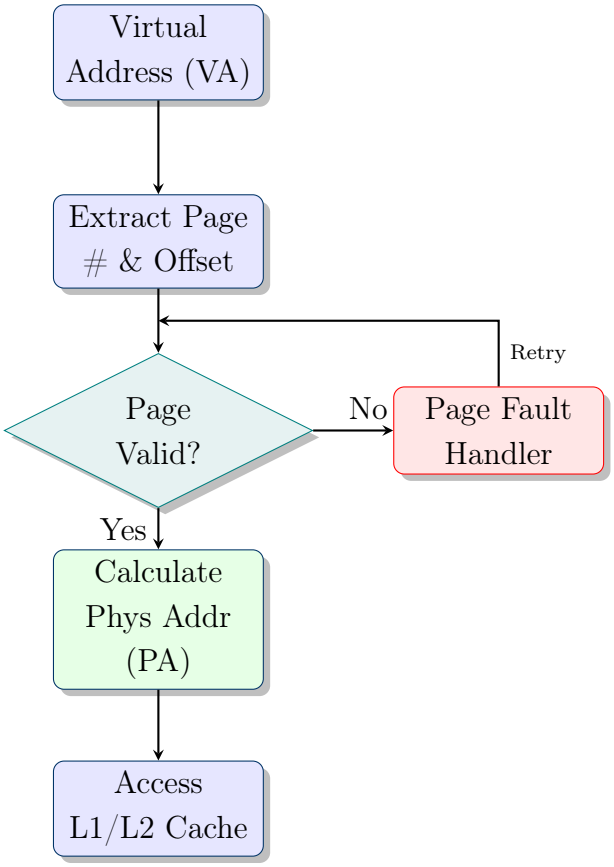

Figure 2: Address Translation Flowchart

5.3 Page Replacement

When a Page Fault occurs and memory is full, the system executes a FIFO eviction:

1. Identify the victim page with the smallest `loaded_at` value.
2. Invalidate the victim's entry in the Page Table.
3. Map the new page to the freed frame.

6 Limitations and Simplifications

1. **Static Sizing:** Total memory and page sizes are fixed at initialization and cannot grow dynamically.
2. **Process Isolation:** The simulator assumes a single process; there is no context switching or multiple page tables.