Design and Implementation of a Memory Management Simulator

Overview

Memory management is a fundamental responsibility of an operating system, involving the allocation, deallocation, caching, and virtualization of memory resources. This project focuses on implementing a **memory management simulator** that models how an OS manages physical and virtual memory.

The simulator will support multiple **dynamic memory allocation strategies**, simulate **multilevel CPU caches**, and implement **virtual memory** using paging. The goal is not to build a real OS kernel, but to **accurately model and simulate OS memory-management behavior** using well-defined data structures and algorithms.

This project emphasizes algorithmic correctness, performance trade-offs, and systems-level design.

Objectives

The primary objectives of the project are:

- To understand dynamic memory allocation strategies and fragmentation.
- To simulate memory allocation and deallocation at runtime.
- To implement and compare cache replacement policies.
- To model virtual memory using paging and address translation.
- To gain hands-on experience with OS-level abstractions using user-space code.

Project Features

1. Physical Memory Simulation

- Simulate a contiguous block of physical memory of configurable size.
- Memory is divided dynamically based on allocation requests.
- Memory units may be represented as bytes, words, or fixed-size blocks (must be documented).

2. Memory Allocation Strategies

Implement the following allocation algorithms:

- **First Fit**
- **Best Fit**
- **Worst Fit**

Each allocation request must:

- Find a suitable free memory block.
- Allocate memory by splitting blocks if necessary.
- Track allocated and free blocks explicitly.

Each deallocation request must:

- Free the specified memory block.
- Coalesce adjacent free blocks to reduce fragmentation.

3. Allocation Interface

Provide a command interface supporting:

- `malloc(size)`
- `free(address or block_id)`
- Memory dump / visualization command showing:
  - Allocated blocks
  - Free blocks
  - Fragmentation statistics

An example of a CLI based interface is:

```
$ ./memsim


> init memory 1024

> set allocator first_fit


> malloc 100

Allocated block id=1 at address=0x0000


> malloc 200

Allocated block id=2 at address=0x0064


> free 1

Block 1 freed and merged


> dump memory

[0x0000 - 0x0063] FREE

[0x0064 - 0x012B] USED (id=2)

[0x012C - 0x03FF] FREE


> stats

Total memory: 1024

Used memory: 200
```

External fragmentation: 35%

4. Metrics and Statistics

The simulator must compute and report:

- Internal fragmentation
- External fragmentation
- Allocation success/failure rate
- Memory utilization

5. Buddy Allocation System (Optional)

Implement a **Buddy Memory Allocation** scheme:

- Memory size must be a power of two.
- Allocation rounds requested sizes up to the nearest power of two.
- Maintain free lists for each block size.
- Support recursive splitting and buddy coalescing.

6. Multilevel Cache Simulation

Simulate a **multilevel CPU cache hierarchy**, such as:

- L1 Cache
- L2 Cache
- (Optional) L3 Cache

Each cache level must have:

- Configurable size
- Block size
- Associativity (direct-mapped or set-associative)

Cache Replacement Policies

- **FIFO (First-Come, First-Served)**

Optional:

- LRU (Least Recently Used)
- LFU (Least Frequently Used)

Track and report:

- Cache hits and misses per level
- Hit ratio
- Miss penalty propagation to lower cache levels

7. Virtual Memory Simulation (Optional)

Implement a virtual memory system using **paging**.

Address Translation

- Define:
    - Virtual address size
    - Page size
    - Physical memory size
- Implement page tables for each simulated process.
- Translate virtual addresses to physical addresses.

Page Replacement Policies

Implement one or more of the following:

- FIFO
- LRU
- Clock (optional extension)

On page fault:

- Select a victim page
- Evict it from physical memory
- Load the required page

Page Fault Handling

- Track:
  - Page faults
  - Page hits
- Simulate disk access latency (optional, symbolic).

8. Integration Between Components (if you are implementing Virtual Memory)

- Cache accesses should occur **after address translation**.
- Virtual memory accesses should go through:

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

- Clearly document the order of operations.

Deliverables

1. Source Code

- Complete, documented source code.
- Modular structure separating:
  - Allocation algorithms
  - Cache simulation
  - Virtual memory
  - Statistics and reporting
- Build instructions (Makefile / CMake / setup script).

2. Design Document

A written document describing:

- Memory layout and assumptions

- Allocation strategy implementations
- Buddy system design
- Cache hierarchy and replacement policy
- Virtual memory model
- Address translation flow
- Limitations and simplifications

Include diagrams where appropriate.

3. Test Artifacts

- Input workloads simulating memory allocation patterns.
- Cache access logs.
- Virtual address access logs.
- Expected outputs or correctness criteria.
- Test Scripts (which may be automated as well)

4. Demonstration Materials

- Screenshots or logs showing:
  - Allocation/deallocation behavior
  - Cache hits/misses
  - Page faults
- Optional demo video or terminal recording.

Hints (To Get Started)

1. Project Layout

Recommended repository structure:

```
memory-simulator/
├─ src/
│  ├─ allocator/
│  ├─ buddy/
│   ├─ cache/
│   ├─ virtual_memory/
│   └─ main.cpp
├─ include/
├─ tests/
├─ docs/
└─ Makefile
```

2. Allocation Algorithms

- Represent memory blocks using:
    - Linked lists
    - Explicit block headers
- Always coalesce free blocks after deallocation.
- Track fragmentation metrics explicitly.

3. Buddy Allocation

- Use arrays or maps of free lists indexed by block size.
- Buddy address can be computed using XOR operations.
- Keep block sizes and memory base addresses aligned.

4. Cache Simulation

- Model cache lines explicitly.
- For LRU:
    - Use counters, timestamps, or linked lists.
- For FIFO:
    - Maintain insertion order.

5. Virtual Memory

- Use simple page tables (arrays or maps).
- Page table entries must track:
    - Valid bit
    - Frame number
    - Replacement metadata
- Keep translation logic independent from caching.

6. Common Pitfalls

- Forgetting to merge free blocks.
- Mixing cache logic with memory allocation logic.
- Incorrect address calculations during paging.
- Not defining clear simulation assumptions.

Resources

Operating Systems

- *Operating System Concepts* – Silberschatz, Galvin, Gagne
- *Modern Operating Systems* – Andrew Tanenbaum
- Youtube Playlist on Operating Systems – Gate Smashers - https://youtube.com/playlist?list=PLxCzCOWd7aiGz9donHRrE9I3Mwn6XdP8p&si=rKJkKWOQ140nxyx1
- GeeksForGeeks Blogs - https://www.geeksforgeeks.org/operating-systems/operating-systems/

Evaluation Criteria (Suggested)

- Correctness of allocation/deallocation
- Faithful implementation of algorithms
- Quality of abstractions and modularity

- Clarity of documentation
- Depth of implemented extensions