Memory Management Simulator

Memory Layout and Assumptions

- **Contiguous Memory:** RAM is modeled as a flat, contiguous array to simplify index-based physical addressing.
- **Power of Two Alignment:** We assume total memory size is a power of 2 (2^{10} = 1024) to support the Buddy System's binary tree splitting logic.
- **No Internal Fragmentation in Sequential Fits**: We assume First, Best, and Worst fit allocate exactly the requested bytes, whereas Buddy System accepts internal fragmentation for faster coalescing.
- **Static Page Table:** The Page Table size is determined at initialization and does not grow dynamically, assuming a fixed virtual address space.

Allocation Strategy Implementations

First-Fit:

- Allocates the first available memory block that is large enough to satisfy the request. This is a simple and fast algorithm, but it can lead to fragmentation.

Pros:

- Simple to implement
- Fast allocation

Cons:

- Can lead to external fragmentation

Best-Fit:

- Allocates the smallest available memory block that is large enough to satisfy the request, reduce fragmentation, but it requires searching the entire memory space.

Pros:

- Reduces external fragmentation
- Efficient memory utilization

Cons:

- Slower allocation due to searching

Worst-Fit:

- Allocates the largest available memory block to satisfy the request. The idea is to leave larger contiguous blocks for later use. It may seem counter-intuitive.

Pros:

- Attempts to reduce fragmentation
- Can improve large allocation success rate

Cons:

- Can lead to less efficient memory use

Next-Fit:

- Similar to First-Fit, but it starts searching for an available block from the last allocated block's location. This aims to distribute allocations more evenly.

Pros:

- Faster than First-Fit in some cases.
- More even distribution than First-Fit.

Cons:

- Performance can vary depending on the allocation pattern.

Cache Hierarchy and Replacement Policy

1. **L1 Cache**: Small, high-speed (4 slots), Direct-Mapped.
2. **L2 Cache**: Larger (8 slots), Direct-Mapped, acting as a secondary buffer before RAM.
3. **Mapping**: Physical Address % Number of Slots.
4. **Write Policy**: Write-through (data is updated in RAM immediately).

```
> init 1024
Memory: 1024 bytes.
> init_vm 5
Virtual Memory ready.
> init_cache 4 8
L1/L2 Caches initialized.
> set allocator buddy
Policy updated to: buddy
> malloc 100
Buddy System: Rounding 100 to 128
Allocated (Best Fit) block id=1 at address 0
> malloc 100
Buddy System: Rounding 100 to 128
Allocated (Best Fit) block id=2 at address 128
> v_access 0
PAGE FAULT! Page 0 not in RAM.
Page 0 loaded into RAM.
Translation: Virtual 0 -> Physical 128
Accessing Physical Address 128...
L1 Miss. Checking L2...
L2 Miss. Fetching from Main Memory.
> v_access 0
Translation: Virtual 0 -> Physical 128
Accessing Physical Address 128...
L1 Hit!
> v_access 64
PAGE FAULT! Page 1 not in RAM.
Page 1 loaded into RAM.
Translation: Virtual 64 -> Physical 192
Accessing Physical Address 192...
L1 Miss. Checking L2...
L2 Miss. Fetching from Main Memory.
> v_access 128
PAGE FAULT! Page 2 not in RAM.
```

```
Page 2 loaded into RAM.
Translation: Virtual 128 -> Physical 256
Accessing Physical Address 256...
L1 Miss. Checking L2...
L2 Miss. Fetching from Main Memory.
> v_access 192
PAGE FAULT! Page 3 not in RAM.
Page 3 loaded into RAM.
Translation: Virtual 192 -> Physical 320
Accessing Physical Address 320...
L1 Miss. Checking L2...
L2 Miss. Fetching from Main Memory.
> v_access 256
PAGE FAULT! Page 4 not in RAM.
RAM full. Evicted Page 0
Page 4 loaded into RAM.
Translation: Virtual 256 -> Physical 384
Accessing Physical Address 384...
L1 Miss. Checking L2...
L2 Miss. Fetching from Main Memory.
> free 1
Block 1 freed.
> free 2
Block 2 freed.
> dump

--- Memory Map ---
[0 - 1023] FREE
------------------

> report

--- Memory Statistics ---
Utilization: 0%
External Fragmentation: 0%
Estimated Internal Fragmentation: ~10% (Power of 2 padding)
-------------------------

L1 Miss Rate: 83.3333%
>
```

Buddy System Design: Memory is treated as a single block of 2^N. Requests are rounded up to the nearest power of 2; if a large block is split, it creates "buddies". When freed,

buddies merge (coalesce) only if both are free and were split from the same parent.

Virtual Memory Model & Translation Flow

- **Model**: The simulator implements a **Demand-Paging** system where physical memory frames are allocated only upon a request for a virtual address that is not currently mapped.
- **Page Size**: Fixed at 64 bytes. This means for any virtual address, the lower 6 bits function as the **Offset** (2^6 = 64), and the remaining upper bits represent the **Page Number**.

🎯Logic:

1. **Address Splitting**: The system receives a Virtual Address and mathematically extracts the page index (Address / 64) and the offset (Address % 64).
2. **Page Table Lookup**:
   a. If the Page Table entry is **Valid**, it retrieves the assigned **Physical Frame Number**.
   b. If **Invalid**, a **Page Fault** is triggered.
3. **Victim Selection (FIFO)**:
   a. If all physical frames are occupied, the system looks at the **FIFO Queue**.
   b. The page that has been in RAM the longest (the "front" of the queue) is evicted to make room for the new page.
4. **Physical Address Generation**: The Physical Address is calculated as (Frame Number \times 64) + Offset.

Cache Hierarchy and Replacement Policy

1. Multi-Level Architecture

- **L1 Cache**: A small, high-speed buffer (default 4 slots) that provides the first point of check for any memory request.
- **L2 Cache**: A larger, secondary buffer (default 8 slots) that serves as a bridge between L1 and the much slower Main Memory (RAM).

2. Placement Policy (Direct-Mapping)

- To ensure constant-time $O(1)$ lookups, both cache levels use **Direct-Mapping**.
- Each physical address is mapped to exactly one possible slot using the formula: Index = (Physical Address / Word Size) \mod Number of Slots.

- This simulates the simplest form of cache management found in hardware, where the hardware logic uses bits from the address to determine the storage location.

3. Replacement Policy

- Because the mapping is direct, replacement is deterministic.
- If a new address maps to a slot that is already occupied by a different address, the existing data is automatically evicted and replaced by the incoming data.
- This ensures the cache always contains the most recently accessed data for a specific set of addresses (temporal locality).

4. Write Policy

- The simulator implements a **Write-Through** policy.
- Whenever data is written to the cache, it is immediately updated in the Physical RAM as well to ensure data consistency across all levels.

5. Data Propagation Logic

- **On a Hit**: The data is returned immediately from L1 or L2.
- **On a Miss**: The system fetches the data from the next slower level (L1 to L2 to RAM).
- **Backfilling**: Once data is retrieved from RAM, it is "pushed" into both L1 and L2 caches so that subsequent accesses to the same address result in an L1 Hit."

Limitations and Simplifications

- **No Multi-threading**: The simulator operates in a single-threaded environment, meaning no mutexes, semaphores, or locks are implemented. In a production OS, memory allocation must be thread-safe to prevent race conditions during block splitting or coalescing.
- **Disk Simulation (Swap Space)**: When the FIFO replacement policy selects a victim, evicted pages are simply "discarded" rather than written to a simulated swap-file on a disk. Consequently, any "dirty" data in those pages is lost, and the system assumes the backing store is always available to reload the page on the next fault.
- **Fixed Page/Frame Size**: For architectural simplicity, the page size is hardcoded to 64 bytes. Real systems often support multiple page sizes (e.g., 4KB, 2MB "huge pages") to reduce Page Table overhead for large applications.
- **Write-Through Cache Only**: The cache hierarchy uses a Write-Through policy where every write to L1 is immediately propagated to RAM. While this ensures data

consistency, it does not simulate the performance benefits of a "Write-Back" policy, which reduces memory bus traffic.

- **Sequential Scan Overhead**: For First-Fit, Best-Fit, and Worst-Fit, the simulator uses a linear search (O(N)) through the block list. A real-world allocator would likely use indexed trees or "size-segregated" lists to achieve faster allocation times.