Project 3 Report: User-level Memory Management

CS 416/518: Operating Systems Design

Team Members:

- Kelvin Ihezue (NetID: ki120)
- Bryan Shangguan (NetID: bys8)

---

1. Detailed Implementation Logic (Virtual Memory Functions)

Refers to the 32-bit implementation in my_vm.c and my_vm.h.

Initialization (set_physical_mem)

- Allocates a 1GB buffer (g_physical_mem) using malloc to simulate physical RAM.
- Initializes two bitmaps: g_phys_bitmap to track physical frame allocation and g_virt_bitmap to track virtual page allocation.
- Allocates the top-level Page Directory.
- Uses pthread_mutex locks to ensure thread safety during initialization.

Address Translation (translate)

- **TLB Lookup:** First checks the TLB using TLB_check. If valid, returns the cached physical address immediately.
- **Page Walk:** If TLB misses, extracts the Page Directory Index (PDX) and Page Table Index (PTX) using bit masking/shifting macros defined in my_vm.h.
- **Two-Level Traversal:**
  - Checks if the PDE is present.
  - Calculates the address of the inner Page Table.
  - Checks if the PTE is present.
- **TLB Update:** If the translation is found in the page table, it adds the entry to the TLB.

Page Mapping (map_page)

- Takes a virtual address and physical address as input.
- Walks the Page Directory. If a Page Table is missing for the required directory entry, it finds a free physical frame, allocates a new Page Table, and updates the PDE.
- Updates the specific Page Table Entry (PTE) with the Physical Frame Number (PFN) and sets status bits (PTE_PRESENT, PTE_WRITABLE).
- Uses phys_bitmap_lock to ensure two threads do not grab the same free frame simultaneously.

Memory Allocation (n_malloc)

- **Size Calculation:** Rounds up the requested bytes to the nearest number of full pages.
- **Virtual Allocation:** Uses get_next_avail to scan the virtual bitmap (g_virt_bitmap) for a contiguous sequence of free virtual pages.
- **Physical Allocation:** Iterates through the required number of pages. For each, it finds a free physical frame using the physical bitmap.
- **Mapping:** Calls map_page to link the virtual page to the physical frame.
- **Error Handling:** If allocation fails at any step (e.g., out of physical memory), it performs a rollback, freeing any bits or pages set during the attempt.

Memory Deallocation (n_free)

- Calculates the number of pages to free based on the size argument.
- **Bitmap Cleanup:** Clears the corresponding bits in g_virt_bitmap to release the virtual address space.
- **Page Table Cleanup:** Walks the page table for the specified range. For every valid PTE found:
    - Extracts the PFN.
    - Clears the corresponding bit in g_phys_bitmap.
    - Invalidates the PTE (sets it to 0).

Data Movement (put_data / get_data)

- **Boundary Handling:** Handles read/write operations that span multiple pages. It calculates how much data fits in the current page (PGSIZE - offset) and copies data in chunks.
- **Translation:** Calls translate for every page accessed to ensure the physical address is valid before copying.
- **Memcpy:** Uses memcpy to transfer data between the user buffer and the simulated physical memory (g_physical_mem).

TLB Implementation (add_TLB, check_TLB)

- **Structure:** Implements a direct-mapped cache.
- **Indexing:** Uses modulo (vpn % TLB_ENTRIES) to determine the index.
- **Eviction:** Because it is direct-mapped, collisions simply overwrite the existing entry at that index (compulsory eviction).
- **Miss Rate:** Tracks total lookups and misses to calculate the rate in print_TLB_missrate.

---

2. Benchmark Output (Part 1 & 2)

- **Matrix Multiplication Result:** Success/Verified.
- **TLB Statistics:**
    - Total Lookups: 403

- Total Misses: 3
- TLB Miss Rate: 0.007444

---

3. Support for Different Page Sizes

- **Macro-based Design:** The implementation relies on constants defined in my_vm.h (PGSIZE, MAX_MEMSIZE, PT_ENTRIES, PD_ENTRIES).
- **Bit Manipulation:** Address translation logic uses OFFBITS, PDXSHIFT, and PTXSHIFT. Changing the PGSIZE definition automatically adjusts the bit masks and shift values, allowing the code to support 8KB or larger pages without logic changes.
- **Linear Scaling:** The bitmap sizes and malloc calculations dynamically adjust based on MEMSIZE / PGSIZE (defined as NUM_PHYS_FRAMES), ensuring the bitmap covers all frames regardless of page size.

---

4. Possible Issues

- **Internal Fragmentation:** n_malloc allocates full pages even for small requests (e.g., 1 byte requests consume 4KB). This leads to wasted space within pages.
- **Search Latency:** get_next_avail performs a linear search on the virtual bitmap. As memory fills up, finding a large contiguous block of free virtual memory becomes slower O(N).
- **TLB Thrashing:** Since the TLB is direct-mapped, alternating access between two pages that map to the same TLB index will cause constant evictions and a high miss rate.

---

5. Extra Credit: 64-bit 4-Level Page Table

Refers to implementation in my_vm64.c.

Design Changes

- **4-Level Hierarchy:** Implements PML4 → PDPT → Page Directory → Page Table. This supports the larger virtual address space required for 64-bit systems.
- **Data Types:** All address variables, page table entries (pte_t), and directory entries (pde_t) are updated to uint64_t.
- **Virtual Address Space:** Supports 48-bit virtual addressing (1ULL << 48) as specified in the requirements.

VMA Management (Linked List)

- Unlike the 32-bit bitmap approach, the 64-bit implementation uses a linked list (struct vm_area) to manage virtual memory regions.
- **Nodes:** Each node tracks a start address and size.

- **Allocation:** get_next_avail traverses the sorted linked list to find a "gap" between allocations large enough to fit the requested number of pages. This is more efficient for the sparse 64-bit address space than a massive bitmap.

Logic Adjustments

- **Translation:** The translate and map_page functions now cascade through 4 levels of translation. If any intermediate table (PML4, PDPT, PD) is missing during mapping, a new physical frame is allocated and zeroed out to create that table.
- **TLB:** The TLB struct was updated to store 64-bit VPNs and PFNs.
- **Thread Safety:** Maintains granular locking with phys_bitmap_lock (frames), vm_lock (VMA list), pt_lock (page tables), and tlb_lock (cache).