MP3: Page Manager

The PageTable class implements a basic and efficient paging system for an operating system. This report outlines the structure, implementation, and functionality of the page manager using the design and code found in page_table.h and page_table.cpp. By adhering to the guidelines and principles from these files, I was able to create a concise solution for managing page tables, handling page faults, and ensuring smooth logical-to-physical memory mapping. The ContFramePool class is used to allocate frames for page directories and page tables, making it a crucial part of the memory management process.

page_table.h

The header file for the PageTable class was structured to declare the key components of the paging system, including the page directory, frame pools, and essential functions for paging. I did not make alterations to this file.

PageTable Class: The PageTable class manages the translation between logical and physical memory addresses by using page directories and page tables. It contains the following.
<u>Static Members:</u>

- **current_page_table**: Points to the currently loaded PageTable object.
- **paging_enabled**: Indicates whether paging is turned on (logical memory addressing).
- **kernel_mem_pool**: A frame pool for kernel memory.
- **process_mem_pool**: A frame pool for process memory.
- **shared_size**: The size of the shared address space, which is crucial for managing shared memory between processes.

<u>Instance Members:</u>

- **page_directory**: Points to the physical location of the page directory for the current page table.

<u>Constants</u>

- **PAGE_SIZE:** Defines the size of each page (in bytes), as provided by the underlying hardware (Machine::PAGE_SIZE).
- **ENTRIES_PER_PAGE:** Specifies the number of entries in a page table (Machine::PT_ENTRIES_PER_PAGE).

Functions:

init_paging: Sets up the global paging parameters, including the kernel and process memory pools, and shared memory size.

PageTable(): Constructor that initializes the page table by allocating a page directory and setting up the first page table for the first 4MB of memory.

load(): Loads the current page table into the CPU's control register (CR3) to switch address spaces.

enable_paging(): Enables paging by setting the paging flag in the control register (CR0). This allows the CPU to translate logical addresses into physical addresses.

handle_fault(): Handles page faults by allocating new page tables or frames when a logical address is accessed that does not yet have a mapped physical page. This is critical for demand paging.

page_table.cpp

The implementation of the PageTable class provides the functional aspects of paging, such as allocating page tables, mapping memory, and handling page faults. Below are the key points of the implementation:

Static Initialization
Static members, including current_page_table, paging_enabled, kernel_mem_pool, and process_mem_pool, are initialized to nullptr and 0 to establish the basic environment before paging is turned on.

init_paging()
This function sets up the global parameters required for paging. It links the kernel and process frame pools and sets the shared memory size. This provides the foundation for paging across both kernel and process memory spaces.

Constructor: PageTable()

- Allocates a frame from the kernel memory pool to serve as the page directory.
- Maps the first 4MB of memory by creating and populating the first page table.
    - This involves setting the page directory entry to point to the first page table and marking the pages as present, supervisor-level, and read/write (using bitwise OR with 0x3).
- The constructor also supports recursive mapping, where the last page directory entry points to the page directory itself. This facilitates easier management of page tables by mapping the page directory into the logical address space.

load()

- This function writes the page directory's address into the CR3 register, effectively loading the page table for use by the CPU. This function must be called when switching address spaces (e.g., during context switching).

enable_paging()

- Paging is enabled by modifying the CR0 register, setting the most significant bit to 1. This activates logical memory addressing, allowing the CPU to manage memory using the page tables.

handle_fault()

- Reads the faulting address (stored in CR2) and extracts the page directory and page table indices.
- Checks if the page directory entry corresponding to the faulting address is present. If not, a new page table is allocated from the kernel memory pool, and its address is inserted into the page directory.
- Checks if the page is present in the page table. If not, a new frame is allocated from the process memory pool, and the page table entry is updated with the frame’s address, marking it as present and writable.
- Console output statements provide detailed feedback during the fault handling process, displaying the status of the page directory and page table entries.

Debugging

Similar to the development of the ContFramePool class, I used extensive debugging and console outputs to monitor the behavior of the PageTable class. By outputting messages during key operations such as page table allocation, page fault handling, and paging enabling, I was able to track the state of the paging system closely. This helped me identify and resolve issues with frame allocation, page table entries, and recursive mapping.