CSCE 611 - MP3 (Design Document)

Page Table Management

Objective:

The objective of this machine problem is to implement a page table object that will be used as a virtual memory system for the kernel. The paging mechanism in this implementation is in accordance with the x86's two level paging. The page table is currently set up for handling a single address space.

Requirements:

1. Manage mapping of direct mapped physical memory for the first 4MB.
2. Manage mapping of memory above 4MB where the logical address is not the same as the physical address .
3. Initialize the paging mechanism on the machine and enable paging.
4. Handle page faults and allocate physical frames.

Design:

init_paging — The init_paging is a static method on the PageTable object and initializes the common information required to start the paging process. It mainly initializes the following.

1. The kernel frame pool that will be used.
2. The process frame pool that will be used.
3. The size of the shared memory space that needs to be managed.

PageTable() constructor — The PageTable constructor will set up the paging correctly before loading and enabling paging in the machine. The PageTable constructor will initialize the Page Directory and Page table entries for the direct

mapped physical memory addresses till the 4MB space. The following is done in the constructor.

1. Get a frame from the kernel_mem_pool, this frame will be used as a page_directory.
2. Similarly, get another frame from the kernel_mem_pool, this frame will be used as a page_table_page.
3. Firstly, assign the reference to the page_table_page to the first entry in the page_directory and mark the page reference entry as valid.
4. Second, initialize all the other entries in the page_directory as not present.
5. Now initialize the page_table_page with the physical frame addresses of the 1024 frames that are between 0 and 4MB. Example: Page address for zeroth page is 0, Second page is 4096, Third page is 8192 and so on.
6. Mark all the 1024 frame entries as valid, this concludes the initial setup of the constructor and the page table.

load — The load operation would load the page_directorie's address into the CR3 register and set the static current page table reference to the current object. The variable page_directory represents the address of the page_directory frame and this is loaded into the CR3 register.

enable_paging — The enable paging method will write the CR0 register's 31st bit to 1. This will enable paging and then the kernel switches from physical to logical addressing.

handle_fault — The handle_fault method handles the page fault exception as set up in the kernel.C file. We get to access the REGS error_code and determine if we have to handle this page fault. The following is implemented in the handle_fault method.

1. First, get the reference to the current page_directory address from the CR3 register. This will help us understand where the page fault originated from.
2. Second, read the fault address from the CR2 register.
3. Based on the format of the fault address, which is | 10 bit | 10 bit | 12 bit |.
    a. First determine the index of the page_directory entry, this is done by bitwise right shift of the address by 22 bits.
    b. Subsequently determine the index of the page_table entry, this can also be achieved by bitwise right shift of the address by 12 bits and then by performing bitwise and operation using the mask 0x3FF.

    For Example:

    If the fault address is 4202496, which is represented in binary as follows:

    00000000010000000010000000000000

    Bitwise right shifting by 22 bits will yield the page_directory_index as 1.

    00000000000000000000000000000001

    Similarly, right shifting the original value by 12 and applying the bitwise 0x3FF will extract the information about the 10 bits which represent the page_index.

    00000000000000000000000000000010 which is 2 in this case.
4. Next, since the error code's last bit represents the present bit, a bit wise xor operation is done to check if the present bit is 0, if it is 0 then we handle the page fault.
5. The page directory entry is checked, to see if the index is valid, if the index is not valid a new page_table_page frame is requested from the mem pool and the page_directory index is updated with the new frame reference, we also mark the page_directory_index as valid.

6. To extract the page_table address from the page_directory_index we apply a mask 0xFFFFF000 which represents the most significant 20 bits as the frame address and filters out the other status flags.
7. The page table entry is checked, to see if the page table index entry is valid, if the index is not valid, we request a new frame from the process frame pool and fill the page_table index with the new frame reference and mark the page_table_index entry as valid.