CSCE 611

Gautham Srinivasan (UIN: 927008557)

MP3 Design Document

Objective:

The objective of machine problem 3 is to design a demand-paging based virtual memory system for our kernel which uses the frame manager of the physical memory to map it to the pages in virtual memory.

Two Level Paging:

The two level paging system consists of page directory which holds the address of 1024 page table and page table which holds the address of 1024 pages. Each address is 32 bit where the bits 31-22 bits (10 bits) are used as a index by the page directory to reference the page table, 21-12 (10 bits) are used as a index by the page table to reference the page and 11-0 bits are the offset which are used to store the state of the pages.

Before proceeding with the paging, we need to initialize the paging system. We request a frame from kernel frame pool for page directory and another frame for page table. The first 4MB of the process is directly mapped to the kernel memory and the rest of the addresses are freely mapped.

Following are the control registers functionality:

CR_0 – When the 31st bit is set, the CPU starts the paging

CR_2 – Address of the page which is marked as not present in the page directory/page table

CR_3 – Used to store address of page directory

a) **init_paging()** – Initializes the parameters i.e kernel frame pool, process frame pool and share_size to set up for paging system.
b) **PageTable()** – Creates page directory and page table by requesting a frame from kernel frame pool and maps first 4MB to the physical address space.

Detailed Implementation:

1. Request a frame from kernel frame pool for page directory
2. Request a frame kernel frame pool for page table
3. Map the first 4MB of virtual address to physical address and mark the pages as present

4. Store the page table in the first index of page directory and mark it as present. And mark rest of the entries of the page directories as not present

c) **load()** – Sets the current page table and stores the page directory in CR_3 register.

d) **enable_paging()** – Sets the 31st bit of CR_0 register to start the paging process.

e) **handle_fault() -** This function will be called whenever the CPU is unable to locate the page or the page table the error code 14. The faulty address is read from CR_2 register. It allocates a page if the page is missing from the page table or a page table if the page table is missing from page directory.

Detailed Implementation:

1. Check for the error code 14 which determines the page fault
2. Get the faulty address from CR_2 register.
3. Get the page directory address from CR_3 register.
4. Check whether the present bit is set for the page directory index (page directory [address >> 22]). If yes, then the page table is missing. Request a frame from kernel pool and allocate it to the page directory index.
5. Find the page table using the page directory. Page table = Page directory [faulty address >> 12 & 0x03FF] where the first 20 bits are referenced as the page table since it is a frame.
6. Check whether the present bit is set for the page table index. If yes, then the page is missing. Request a frame from process pool and allocate it to the page table index.