Achtenberg, Alan

CSCE 410

Fall 2014

9/30/14

MP2

This work represents my individual work, and I have listed the sources that I have consulted below. I have not received assistance that would violate the letter or spirit of the collaboration guidelines for this assignment.

Sources: none

DESIGN

In MP2 we implemented a frame pool and a page table. Our system was a 32 MB machine with the kernel space from 0 to 4 MB and the user space from 4MB to 32MB. There also is a 1MB hole of inaccessible memory starting at 15 MB. The kernel space will be direct mapped from virtual memory to physical and shared between process page tables. User space memory will be managed. Both the frames and the pages will be 4 KB in size.

Frame Pool

The frame pool manages contiguous frames of memory, memory below 4 MB is managed by our kernel frame pool and memory above 4 MB is managed by our process frame pool. Although we have two different frame managers the implementation of each is the same so there will only be one frame manager class. The following is a description of the frame pools implementation.

The constructor initializes the frame pool, starting from the base frame number that is nframes long. The info frame number is the frame that will store the management information of the frame pool. If the frame number is 0 the frame pool will store the pool info in the first available frame of the pool. We chose to store our management information in the form of 2 bitmaps of 1024 bits in size (large enough to map our entire memory 32 MB) but. The first bitmap freeframe\_map, is a bitmap that is marked 1 for used and 0 for free. The freeframe\_map just keeps track of free frames. The second bitmap inaccessible\_map, is bitmap that is marked 1 for inaccessible and 0 for accessible. The reason we must have 2 maps will be apparent later. Upon construction the frame pool adds itself to the end of a simple static singly linked list. The end of the list is named root while the previous frame pool\* is named prev. We added this to the frame pool to allow a search of frame pools in the system.

The Get frame function searches through the frame pools bitmaps to find a frame that is both accessible and available and marks the frame as unavailable, before returning the frame number that was found.

The mark inaccessible function, takes a base frame number and nframes argument that will mark the corresponding frames in the inaccessible\_map as inaccessible. An out of range argument will print an error message before doing nothing by returning.

The release frame function, takes a frame number to mark as free. The release frame function is static, so we must first find the correct frame pool to release from our static list of frame pools. Once we find a frame and it is in a valid range, we use the inaccessible\_map to determine if the frame pool as the rights to release the frame. This is the reason for the use of two bitmaps. If we only used one bit map and marked inaccessible frames as used initially it would prevent the user from accessing frames, but if a user were to call release frame on an inaccessible frame there would be no way to differentiate between an invalid and valid release.

Page Table

The page table maps virtual memory into physical memory by utilizing previously created frame pools. Our paging system is two level, we have a page directory which is a pointer to 1024 pages and a page which is a pointer to 1024 places in memory. For any page table initialization the page directory’s first entry is a page table that points to the first 1024 pages in memory which translates to the first 4 MBs of our kernel. This is how we implement the direct mapping and sharing of the kernel space. The rest of the pages are demand paged, or in other words virtual addresses are mapped to physical addresses when needed, this uses the frame pool get frame function to do so.

The Constructor initializes the page directory with a page table that points to the first 4 MBs as mentioned previously. It marks all the other page tables with the not in memory flag 010.

Before the constructor can be called the init\_paging function must be called. This function sets the kernel\_mem\_pool \*, the process\_mem\_pool \* and the shared size of the kernel memory.

The page table load function sets the current page table to this and then writes the page directory pointer into the Cr3 register.

The page table enable\_paging function sets paging enabled to 1 and then sets the 32nd bit of the Cr0 register to 1, this enables the MMU on the hardware to do paging.

The handle\_fault function is a page fault handler, that determines what to do when an access of a page not in memory is made. First the fault handler reads the page directory from the Cr3 register, gaining access. Second it reads the desired access address from the Cr2 register. Third it checks the error code of the handler to determine if we indeed have a not in memory fault. After doing these initial things, we then determine by checking the flag bits in our page directory if the page table corresponding to our address is in memory. If it is in memory we simply create a new page entry in the page table by calling the process mem pool get frame function and loading it into the page table index of our address. If it is NOT in memory we first create a new page table entry in our page directory by calling the kernel mem pool get frame and storing it in the page directory, and then subsequently loading in a new page as previously mentioned. The way the address read from the Cr2 register is interpreted, is the first 10 bits is the index of the page table in the page directory, while the second 10 bits is the index of the page in the page table to be accessed and lastly the last 12 bits is the offset of the pages (4KB) and the flags.