OS611_MP4_Report

Name: Wei-Che Hsu
UIN: 826001725

In this MP, I move page table to process pool, and add the memory pool for virtual address.

First, I would like to talk about the design of page table. The based code is come from my MP3.

In the constructor of page table, I require a physical frame from process memory pool being page directory, and I take the physical address of this frame into CR3 in load function later. Next, I build a directed mapping page table for kernel space (0MB~4MB) with the frames which are from process memory pool. And assign the start address of these PTEs to the first PDE of mentioned page directory. Because I can't use the physical address to find my page directory after paging enable (virtual address and physical address are not directly mapping in the process pool), I take Recursive Page Table Look-up skill to make the last PDE of page directory point to this page directory itself. Besides, because I have to connect virtual memory pool with page table, there is an array for record pools for this page table, and I only set 10 available slots in this array.

In load and enable paging function, these functions are the same as those in MP3.

In handle fault function, I have to check whether the logical address is in the page table's virtual memory pool or not. When I ensure which VMPool it is, and I can also ensure which frame pool it belongs to. Thanks to Recursive Page Table Look-up, I can use 0xFFFFF000 to find the page directory, and use (0xFFC00000 | (PDE_index << 12)) to find the page table from which the reference page cause page fault.

In register pool function, when a virtual memory pool object is created, it will attach to one page table. I use this function to let pool object attach to the page table, and saving this object in the array of virtual memory pool.

In free page function, it will release the requested page which is send from VMPool's function release, and cancel the valid bit of this page.

Second, I would like to talk about the design of virtual memory pool (VMPool). I think the most important thing for VM Pool is the node for recording memory usage. I set these nodes which are saved in the first page of VMPool. For convenience, I will force the head of pool align to page size, and assigned the memory as the multiples of pages.

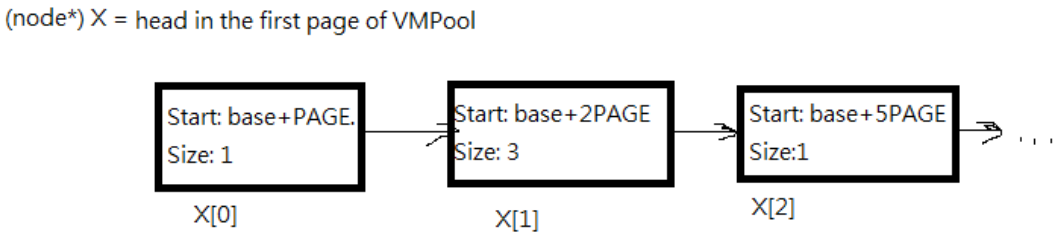

Figure 1: Structure of Allocated_region_node and its array.

Allocated_region_node records start address (unit: Bytes) and size (unit: page). In the first page of virtual memory pool, it can save 512 nodes because the size per node is 8 Bytes. Now, I just use one page to record these nodes, so the maximum number of nodes is 512. That is, An array Allocated_region_node[512] in the first page of VMPool.

In the constructor of VMPool, it just record parameters into its private member variables, and set the first page of this VMPool for Allocated_region_nodes array usage. Also, it calls the page table's register function to register itself to the page table.

In allocate function, the input is required size, and this function should find available space to allocate it. I check whether it can take unused nodes, but the used nodes can't be larger than 512, and try to find whether the hole memory unused in the existing address because when I release the allocated pages, just set the corresponding node's size to 0. There would be some nodes with one start address but with zero size. In any node of these nodes, I can use the start address of this node and the start address of next node to obtain the unused pages in this node. I called it residual pages. If residual pages is large enough to handle request pages, just allocate pages and save info (size and start address) in this node. If there is still unused space in this page, it should add one new size 0 nodes behind this node.

However, if there is no enough residual pages, the allocate function would use one unused node next to the end of used nodes array. Please see the Figure 2 and Figure 3. Finally, this function returns the logical address of allocated pages.

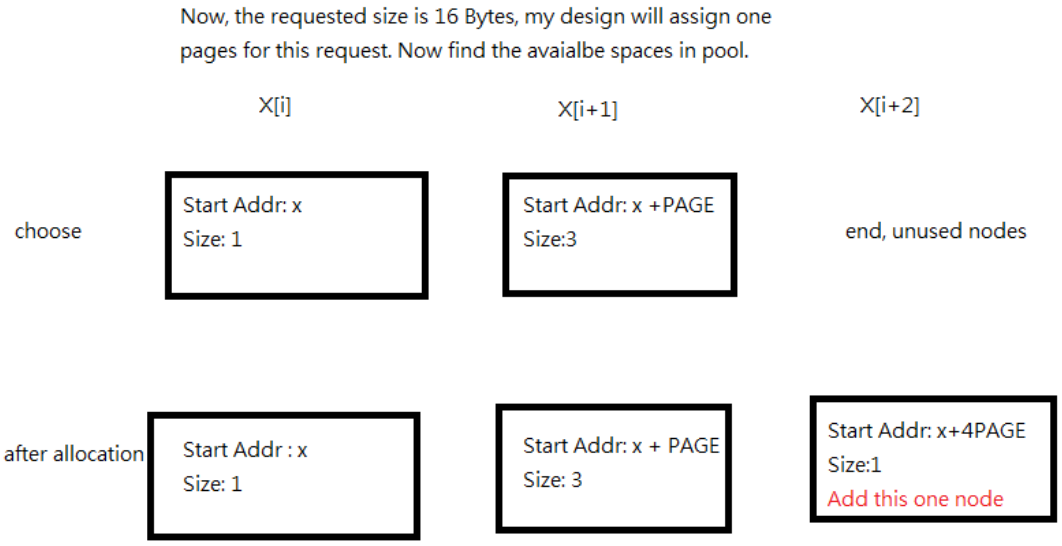

Figure 2: Allocation when there are no enough residual pages

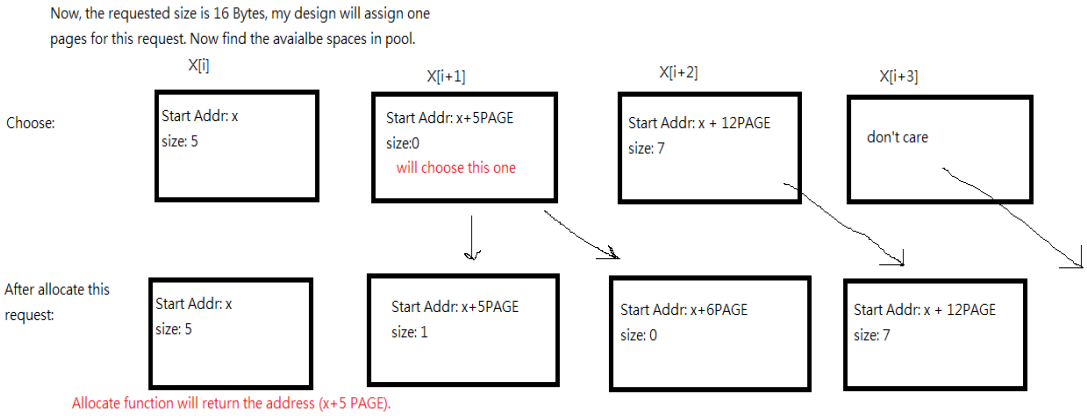

Figure 3: Allocation when there are enough residual pages, and the node in X[i+2] and X[i+3] should do merge check. I would introduce merge action later.

In release function, the input is the released logical address, and this function should let page table set these pages invalid and release corresponding pages. Here, it should also flush TLB after set any page invalid. In VMPool's node, just compare the input logical address to the start address of every node to find the node which is saved the information of the memory usage beginning with input logical address. Release function only set the size in the node to 0, and other task will let page table's function free page to do. However, the noticeable thing is how I reduce the number

of nodes with size 0. If there are many nodes with size0, it should waste these nodes for pool management. Thus, I design a simple method to reduce no-need nodes with size 0. Briefly, I merge the continuous nearby nodes with size 0 after my algorithm make any one node with size 0 each time. Figure 4 and Figure 5 shows merge action for Allocated_region_nodes. Merge action ensures that I would not see the continuous nearby nodes with size 0.

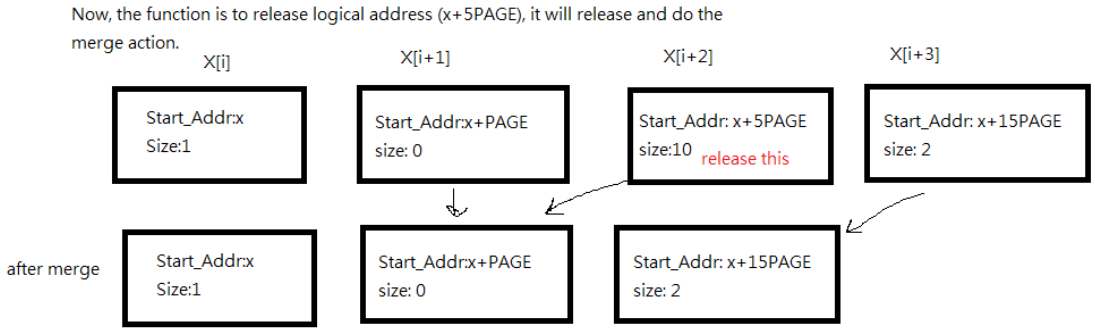

Figure 4: Merge with the previous node with size 0 after releasing

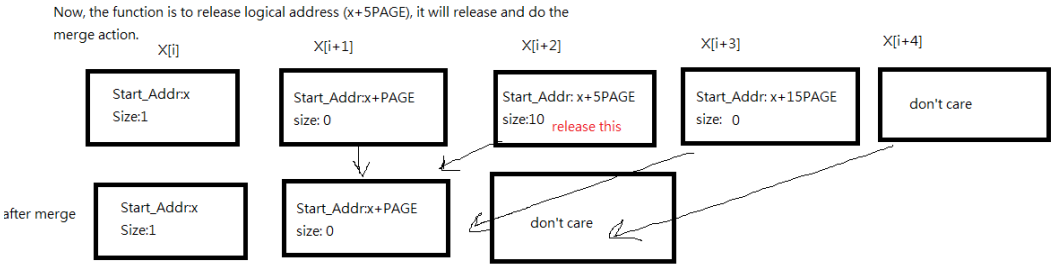

Figure 5: Merge with the previous node with size 0 and the next node with size 0 after releasing

In is_legitimate function, just help page fault handler judge whether pages are in the VMPool or not. Function get_frame_pool reports the frame_pool type to page table function.