MP4: Page Manager 2

Joshua Clapp

UIN: 930004089

CSCE410: Operating System

Assigned Tasks

Main: Completed.

Bonus Option 1:

none.

Bonus Option 2:

none.

Bonus Option 3:

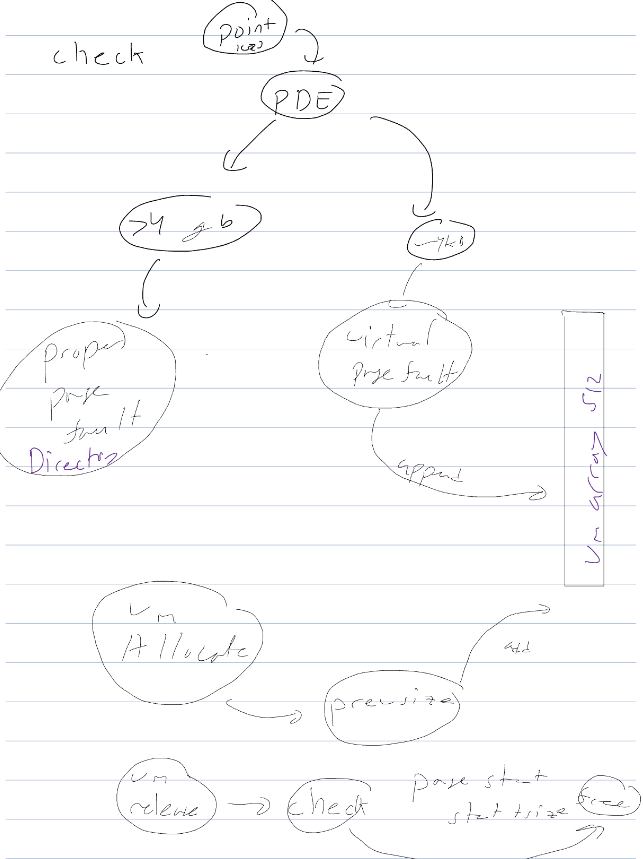
none.

Bonus Option 4:

none.

System Design

* Take the physical address and scale by the page size
  + Handle faults as they come and allocate tables
* Take the VM pool array to hold allocated pages and release quickly.

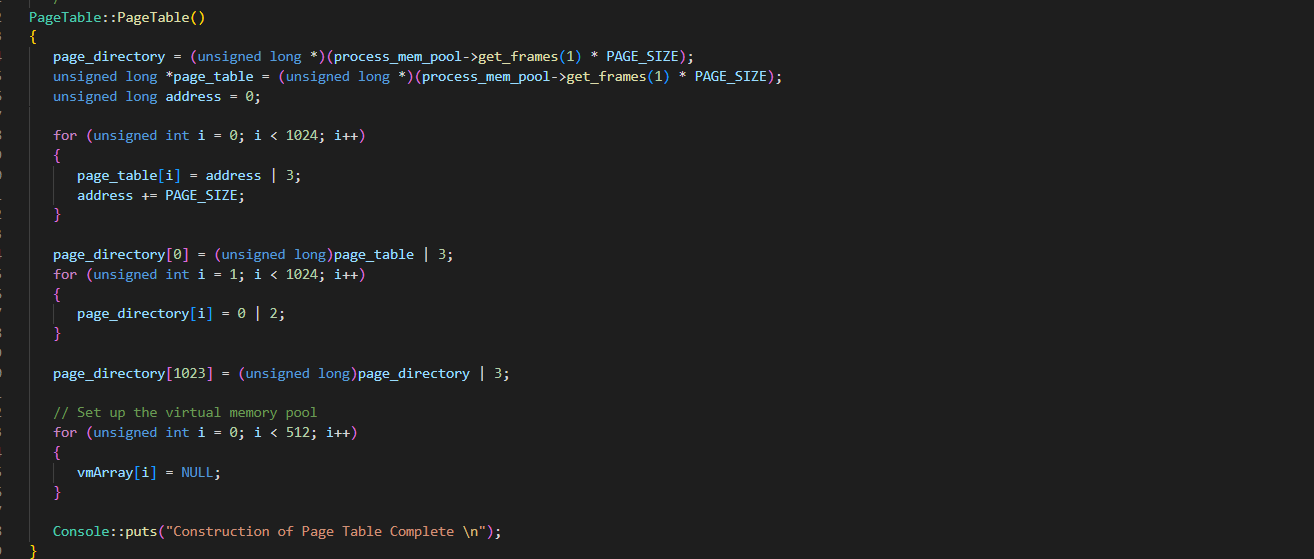


Code Description

I changed page\_table.C, and wrote the Vm\_pool.C to compile the code type make, ./copykernel.sh and then run bochs.

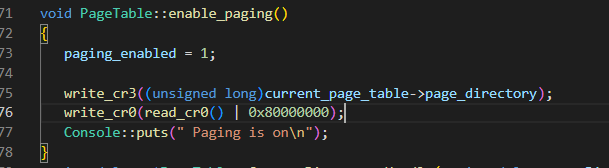
page\_table.C: PageTable():

This method is the same as mp3 where we set the bits, set to user level, set one to supervisor, load a recursive PDE, and then set all of the vm array to nothing.



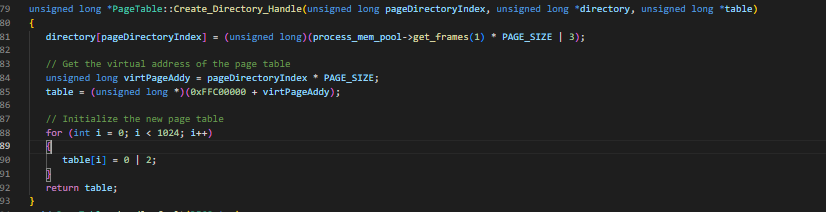
page\_table.C: enable\_paging() :

This is the same method from mp3, besides the pointer to enable paging on the current paging directory.



page\_table.C:Create\_Directory\_Handle():

This is a helper function for the handle fault, it turns on a supervisor directory at 4 gb, and then allocated a page table set to user level. The table is set to X at 12 bits and virtualized.

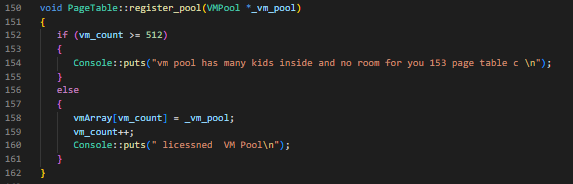


page\_table..C: handle\_fault():

First confirm that there is no standing error code to freak out on, and then check if the vm memory is allocated at the block, and then correctly. Next we check the type of fault whether it is a directory fault at 4gb or a 4kb vm table fault and allocate the frames respectively.

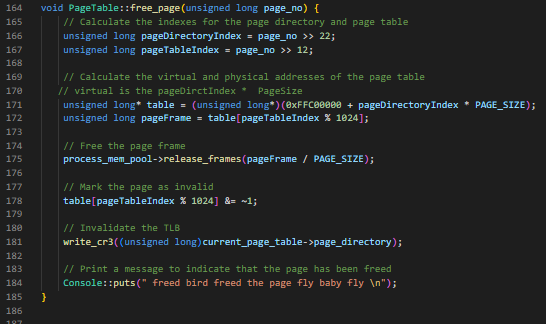


page\_table.C: register\_pool():

Check if the vm\_pool count has hit or exceeded 512, else allocate at that count and increment.

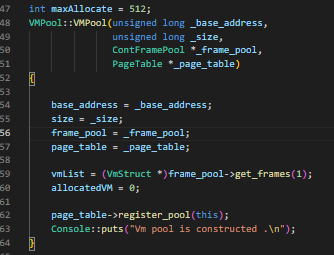
page\_table.C: free\_page():

This function takes the table at the page index of 12 bits ( 4kb ) + 4 gb \* page\_size which is the virtualized bits in the table. Release the frame at the process index from the table accounting for overflow and invert the bits on the table marking it as invalid.



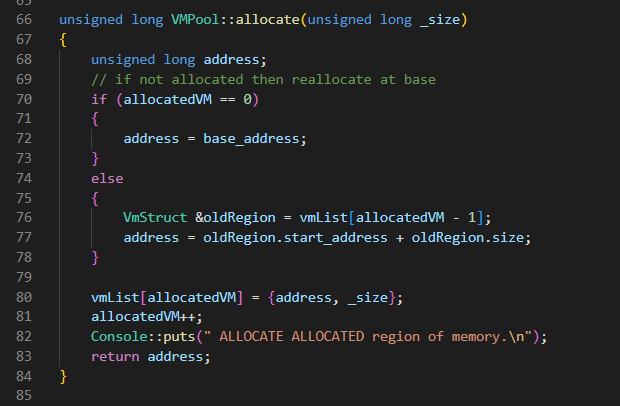
vm\_pool.C: VMPool():

Pass in the active page table, the frame pool, base address, and the size, then allocate the vmList struct to get the necessary frame, and initialize the allocation tracker. Lastly, put the page table into the pool register and begin to track the allocated vm pools.



vm\_pool.C:Allocate():

First, I check if it is the first allocation and set the base address, otherwise the address is the starting region added to the size of the last allocation. Then we create a new allocation for the new position and given size, and increment the allocatedVM count.

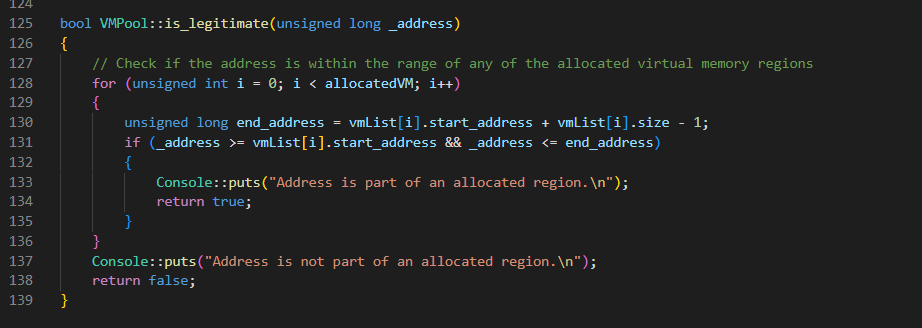


vm\_pool.C:release():

First we check if there is anything to release, once we have found something we free the page table from the physical bit tracker and now shrink the vm pool to reflect the one lesser page table. The last step is to reload the page table and call the updated TLB and virtual memory that this function changed.

vm\_pool.C:is\_legitimate():

This function we check for all allocated pieces of memory if we exceed the ending address on the physical memory / virtual memory boundaries, or don’t meet the given address.



Testing

For testing, I relied on the provided test functions and added nothing. I am not sure how to further test memory allocation and paging. I think the coverage of the testing is comprehensive as it loops through all allocations on a stack and a heap.   
  
Stack and heap testing  
