MP4: Virtual Memory Management and Memory Allocation

Kai-Chih Huang  
UIN: 333000021   
CSCE611: Operating System

Assigned Tasks

Part1: Finished  
Part2: Finished  
Part3: Finished

System Design

The goal of this machine problem is to

Code Description

I changed page\_table.C, page\_table.H, vm\_pool.C and vm\_pool.H for this machine problem. To compile code, simply run following command lines under MP4\_Sources directory:  
  
$ make clean // clean the old compile files before we compile

$ make // compile files

$ ./copykernel.sh (if permission denied, try chmod u+x ./copykernel.sh then do it again) //copy kernel

$ bochs -f bochsrc.bxrc // run bochs  
  
I will walk through the functions/methods defined in above files as follows.

**page\_table.H**

I added a vm pool head pointer to keep track of the linked list of pools.A screenshot of a computer

Description automatically generated

**page\_table.C**

PageTable()  
I changed the frame pool source from the kernel to the process mem pool. When initializing the page directory, I set the last entry to point to the page directory itself.  
A computer screen shot of a program code

Description automatically generated

A computer code with text

Description automatically generated

handle\_fault(REGS \* \_r)  
I set the “recursive address” of the page directory and the faulty page table, and use these to replace the former expression in MP3. I also checked if the faulty address is in one of the registered pool. If not, the system will be terminated.A screen shot of a computer

Description automatically generatedA computer screen shot of text

Description automatically generated

register\_pool(VMPool \* \_vm\_pool)  
If there is no pool, then set the new pool to the vm\_pool\_head. If there is existing pool, we simply walk through the linked list and append the new pool to the tail of the list.A screenshot of a computer program

Description automatically generated

free\_page(unsigned long \_page\_no)  
We use the same way as in handle\_fault(REGS \* \_r) to get the “recursive address” of page table, and the page directory index, page table index as well. If the target PTE is valid, we release the frame by calling release frames() function, then set the PTE to invalid. At the end, we call load() to flush the TLB.A computer screen with text and images

Description automatically generated

**vm\_pool.H**

In addition to the variables to store input parameters, I added some private variables for the VMPool class. Since we need to manage the allocated “regions” in the pool, I defined an AllocatedRegion object to manage the data of 1 region, which includes its base address and size of the memory in Bytes it takes. Then, I use 1 page to store these AllocatedRegion instances, which I can store 256 regions maximum, and I used an array to manage those regions.  
I also added a VMPool pointer to point to the next pool, and total\_regions to show how many regions do we currently have.  
A computer screen shot of a program code

Description automatically generated

**vm\_pool.C**

VMPool(unsigned long \_base\_address, unsigned long \_size, ContFramePool \*\_frame\_pool, PageTable \_page\_table)  
To initialize the VMPool, first we need to register the pool to given page table. Then, we put our array that manage allocated regions in the first page, so the address of the array will be the same as the base address of the pool.  
A computer screen shot of a program code

Description automatically generated

allocate(unsigned long \_size)  
To allocate the frames, it first calculated the required number of frames based on the required memory size. Then, we walk through all the regions and check if there is enough space between them to allocate the requested size. If there is enough space, we insert the new region in between. If not, we add the new region to the end of the array. If there is not enough space between the regions, we add the new region to the end of the array.A screenshot of a computer program

Description automatically generated

release(unsigned long \_start\_address)  
The function release the region that start with given address. Therefore, we simply walk through all the regions and find the region that contains the given address. Then, we free the pages in the region, and shift the regions after the deleted regions to the left by 1.  
A computer screen shot of text

Description automatically generated

is\_legitimate(unsigned long \_address)  
The function checks if the given address is legitimate in this VMPool. Since the first page in the pool is for the allocated\_region\_array, the valid area start from the second page of the pool, and not larger than base\_address + size.A computer screen shot of a code

Description automatically generated

Testing

I did not add additional test data in this MP4, since I believe the provided test set is sufficient to check if the page table system works as expected. The test result is shown as follows.

Test page table  
A screenshot of a computer

Description automatically generated

Test VM pool  
A computer screen shot of a program

Description automatically generated