Document

Zhiren Zhou

October 2022

1 Design Concept

1. Modified the **PageTable::PageTable()** function:

   PageTable::PageTable() function now allocates frames of process memory pool to store the page tables and page directory. The final entry in page directory stores the physical address of page directory, which implements the recursive page table lookup.

2. Added the **PageTable::PDE_address()** function:

   When given a virtual address, this function returns the virtual address of pde corresponding to the given address. Suppose the given address is in form of

   $$pdeIndex(10bit) + pteIndex(10bit) + offset(12bit)$$

   this function should return

   $$1023(10bit) + 1023(10bit) + pdeIndex(10bit) + 00$$

3. Added the**PageTable::PTE_address()** function:

   When given a virtual address, this function returns the virtual address of pte corresponding to the given address. Suppose the given address is in form of

   $$pdeIndex(10bit) + pteIndex(10bit) + offset(12bit)$$

   this function should return

   $$1023(10bit) + pdeIndex(10bit) + pteIndex(10bit) + 00$$

4. Modified the **static PageTable::handle_fault()** function:

   Since the page directory and page tables are stored in the process memory

pool now, they are not in the directly mapped area and this function can not use the physical address to modify the pde and pte in such situation. So we need to modify this function to use the virtual address of pde and pte to make changes to page directory and page tables. To get the virtual address of pde and pte in handler, this function use the two functions mentioned above.

5. Implemented the **PageTable::register_pool()** function:

   This function is used to register a virtual memory pool with the page table. Every page table object has a pointer to the head of registered virtual memory pools list. Whenever a virtual memory pool is registered, it will be added to this list.

6. Implemented the **PageTable::free_page()** function:

   This function is mainly called by **vmpool::release()**. It is used to release frame and mark page invalid. It does not check whether the given page number is valid because this will be checked in **vmpool::release()**.

7. Implemented the **VMPool::VMPool()** function:

   This function constructs a new virtual memory pool. It also allocates the first page of virtual memory pool to store the allocated_list and the freed_list. Whenever a virtual memory object is defined, it will be registered with the current page table object immediately.

8. Implemented the **VMPool::allocate()** function:

   This function looks up the freed list for a region with required size(this size is rounded up to multiple pages). If found, it adds that region to the allocated list and return the start virtual address of that region.

9. Implemented the **VMPool::release()** function:

   This function looks up the allocated list for a region with given base virtual address. If found , it computes the page numbers belonging to that region and releases that region by calling **PageTable::free_page()**.

10. Implemented the **VMPool::is_legitimate()** function:

    This function checks whether an address is valid or not, in other words, whether that address is in an allocated region. One special case is the address that is used to firs time write to the allocated list should always be legitimate. Otherwise, the first page in pool will fail to be written to store the region info. This function will be used to further implement the **static PageTable::handle_fault()**.

11. Modified the **static PageTable::handle_fault()** function:

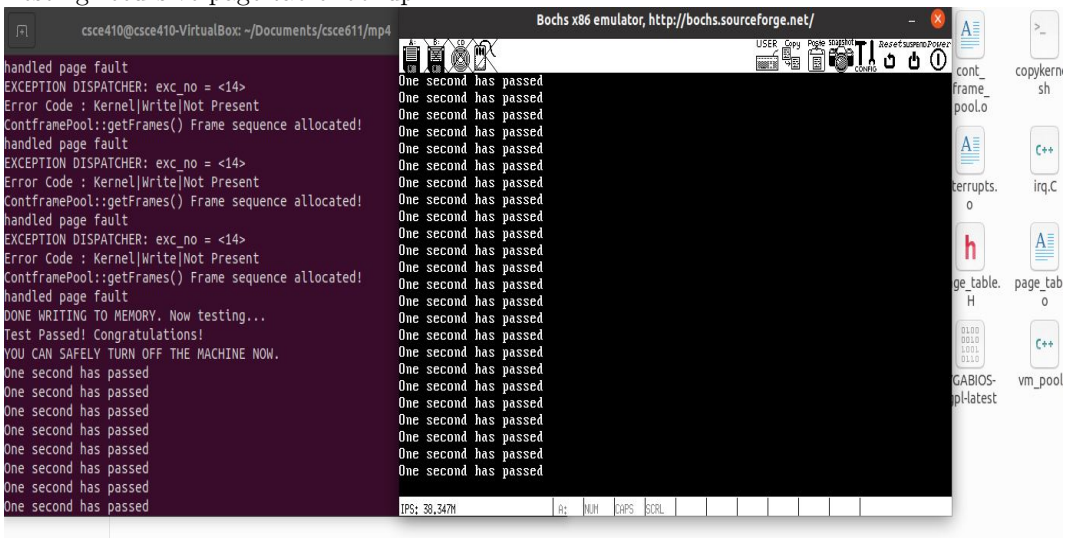
    This function is finally modified to realize a page fault handler for the virtual space with some virtual memory pools. First, this function will check whether the address triggering the fault is a legitimate address by using **VMPool::is_legitimate()**. If not, that is just a mistake reference and the handler does not need to do much thing about it. If yes, the handler will allocates frames from process pool and the modify the page table and directory table to handle the page fault.

2 Files Modified

1. **cont_frame_pool.H**(using implemented one in mp2)
2. **cont_frame_pool.C**(using implemented one in mp2)
3. **page_table.H**
4. **page_table.C**
5. **vm_pool.H**
6. **vm_pool.C**

3 Testing Results

1. Testing recursive page table lookup



2. Testing recursive virtual memory pool