Machine Problem 4: Virtual Memory Management and Memory Allocation

Gaurangi Sinha
UIN: 734004353
CPSC611: Operating System

Assigned Task

Part I: Support for Large Address Spaces Completed.
Part II: Preparing class PageTable to handle Virtual Memory Pools Completed.
Part III: An Allocator for Virtual Memory Completed.

System Design

The objective of this programming assignment was to develop a virtual memory manager capable of managing memory allocations through the new and delete operations. This involved modifying our existing page table mechanism from MP3, transitioning page tables to reside within process space to accommodate extensive address spaces.

- Modifications were made to the page fault handling mechanism to integrate support for virtual memory pools.
- A new class, VMPool, was introduced to oversee the operations of memory pools, including the allocation and deallocation of memory segments.
- We designated the initial page of every new memory pool to hold a structured array. This array records details regarding the memory segments that have been allocated within the pool.

Code Description

I made modifications to various components for managing larger address spaces and implementing virtual memory management in our system. I made several changes in the files "page_table.c", "page_table.h", "vm_pool.c", and "vm_pool.h", using the existing code from "cont_frame_pool.c" and "cont_frame_pool.h" from mp3.

Part I: Enhancing Address Space Capacity with Recursive Page Lookup
To accommodate larger address spaces, the system employs a recursive page lookup strategy. This is implemented by setting the final entry of the page directory to reference the directory itself. The system then navigates through the directory and table entries, we index into the PDE

and PTE values of given address utilizing the format 1023—1023—X—offset and 1023—X—Y—offset.

Part II: Enhancements to the PageTable Class for Virtual Memory Management

The PageTable class underwent significant modifications to manage virtual memory pools effectively. A key addition was the Register Pool API, which keeps a comprehensive list of all memory pools. The page fault handler plays a crucial role here by first verifying the validity of an address, specifically whether it corresponds to an existing memory pool, before proceeding with frame allocation.

Part III: Virtual Memory Allocation with the VMPool Class

The VMPool class introduces critical functionalities for memory management, including an allocation function that searches for available regions to allocate memory segments. It also includes a deallocation function, which ensures the release of memory by interacting with the PageTable class to free up the necessary pages.

PageTable Constructor

```
PageTable::PageTable()
{
    // Initialize a kernel memory pool for managing page tables and directories
page_directory = (unsigned long *) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);
    auto* page_table = (unsigned long *) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);

// Define the initial memory address
unsigned long baseAddr = 0;

// Setup page table entries
for(int idx = 0; idx < 1024; ++idx) {
    // Mark each page as present and read/write
    page_table[idx] = baseAddr | 0x03; // OR operation with 3 sets lowest two bits (present and read/write)
    baseAddr += PAGE_SIZE;
}

// Link page table to the first entry of the page directory
page_directory[0] = (unsigned long) page_table | 0x03;

// Configure the remaining entries of the page directory
for(int dir_idx = 1; dir_idx < 1024; ++dir_idx) {
    // Last entry points to the directory itself, to form a recursive mapping
    if(dir_idx == 1023) {
        page_directory[dir_idx] = (unsigned long) page_directory | 0x03;
    } else {
        // Set pages to supervisor level, read/write, not present
        page_directory[dir_idx] = 0x02; // Only the write bit is set
    }
}

Console::puts("Constructed Page Table object\n");

}
```

Page Fault Handler

```
void PageTable::handle_fault(REGS * _r)
{
        Console::puts("Handling page fault...");
        unsigned int error_code = _r->err_code;

        // Check for protection fault
        if (error_code & 1) {
            Console::puts("Detected Protection Fault - not due to a missing page.");
            return;
        }
        // Fetch the address causing the page fault
        unsigned long fault_addr = read_cr2();

        bool is_pool_page_valid = false;
        int valid_pools_count = 0;

        // Search through VM pools to validate the faulting address
        for (int i = 0; i < 512; i++) {
            if (vm[i] != nullptr) { // Check if VM pool slot is used
                if (vm[i]->is_legitimate(fault_addr)) {
                    is_pool_page_valid = true; // Valid pool page found
                    break;
                } else {
                    ++valid_pools_count;
                }
            }
        }

        // Handling cases with no associated valid pool
        if (!is_pool_page_valid && valid_pools_count > 0) {
            Console::puts("Address does not belong to a valid pool.");
            assert(false);
        }

        // Obtain PDE and PTE pointers for the fault address
        unsigned long *pde = PDE_address(fault_addr);
        unsigned long *pte = PTE_address(fault_addr);

        // Setup PDE and PTE if not already present
        if (*pde & 1) {
            // If PDE is present, setup PTE
            *pte = (unsigned long) (process_mem_pool->get_frames(1) * PAGE_SIZE) | 3;
        } else {
            // Setup both PDE and PTE if PDE is absent
            *pde = (unsigned long) (process_mem_pool->get_frames(1) * PAGE_SIZE) | 3;
            *pte = (unsigned long) (process_mem_pool->get_frames(1) * PAGE_SIZE) | 3;
        }
    Console::puts("handled page fault\n");
}
```

```
void PageTable::register_pool(VMPool * _vm_pool)
{
if(pool_number == 512){
        Console::puts("All VM Pools are in use.");
        assert(false);
    }
    vm[pool_number] = _vm_pool;
    pool_number++;
    Console::puts("registered VM pool\n");
}

void PageTable::free_page(unsigned long _page_no) {
        unsigned long* pte_address = PTE_address(_page_no);

        if (*pte_address & 1) {
        process_mem_pool->release_frames(*pte_address >> 12);
        }
        *pte_address = 0x02;
        write_cr3((unsigned long)page_directory);
        Console::puts("freed page\n");
}
```

```
unsigned long * PageTable::PDE_address(unsigned long addr) {
    unsigned long pde_index = addr >> 20;
    pde_index = pde_index | 0xFFFFF000;
    pde_index = pde_index & ~0x3;
    return (unsigned long*) pde_index;
}

unsigned long * PageTable::PTE_address(unsigned long addr) {
    unsigned long pte_index = addr >> 10;
    pte_index = pte_index | 0xFFC00000;
    pte_index = pte_index & ~0x3;
    return (unsigned long*) pte_index;
}
```

Image above: Function to calculate the pde and pte values from given address

Compilation

To compile the code, we run the following commands:

make clean
make

To run the simulator we run the following command:

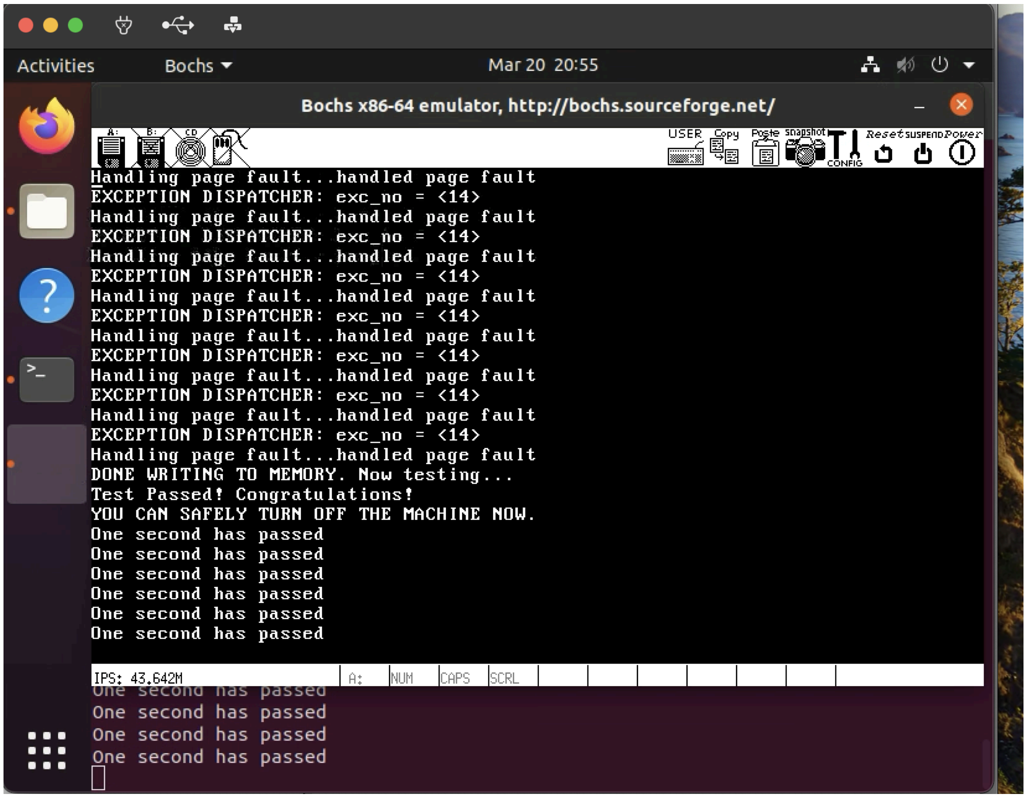
./copykernel.sh

and then to start the Bochs simulator we run the the following command:

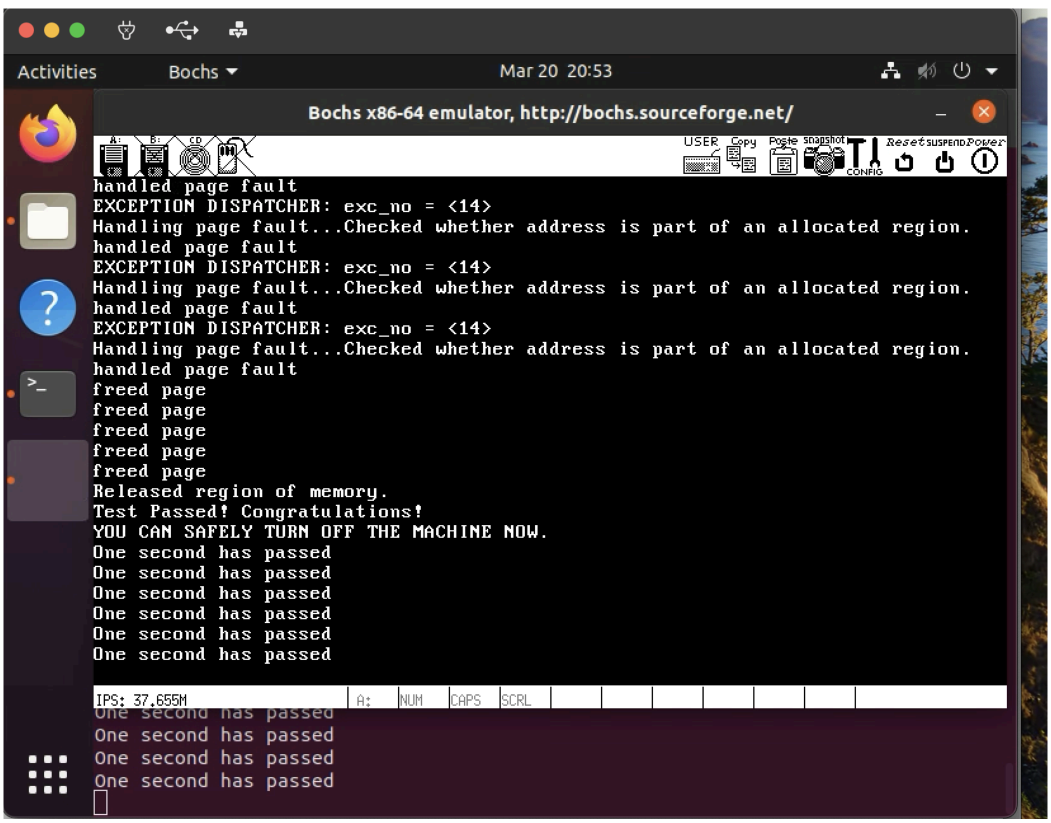
bochs –f bochsrc.bxrc

Testing

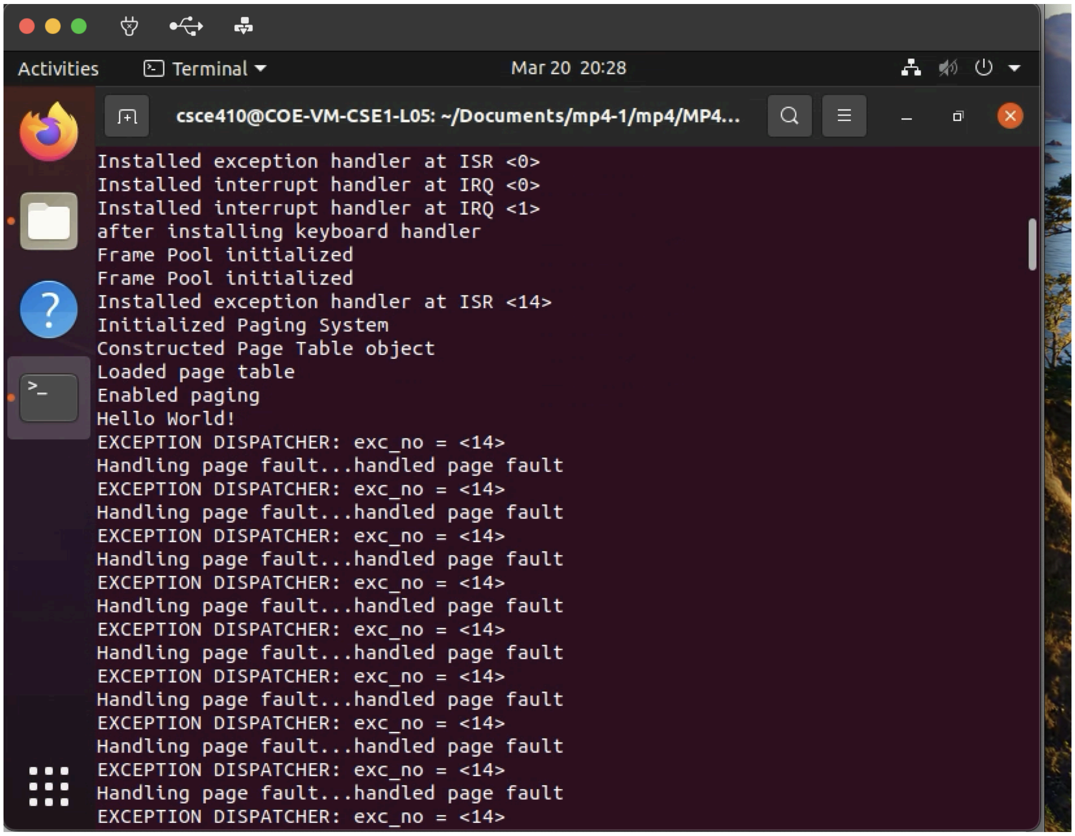
For testing, I used the given test function from kernel.C. I ran two scenarios where the _TEST_PAGE_TABLE macro is set and unset. By default, this macro is defined, and only the page-table implementation is tested. I uncommented the definition of this macro, and the code started testing the VM pools. All the tests are passing, for given and additional scenarios. I have attached the screenshot of the outputs I got.

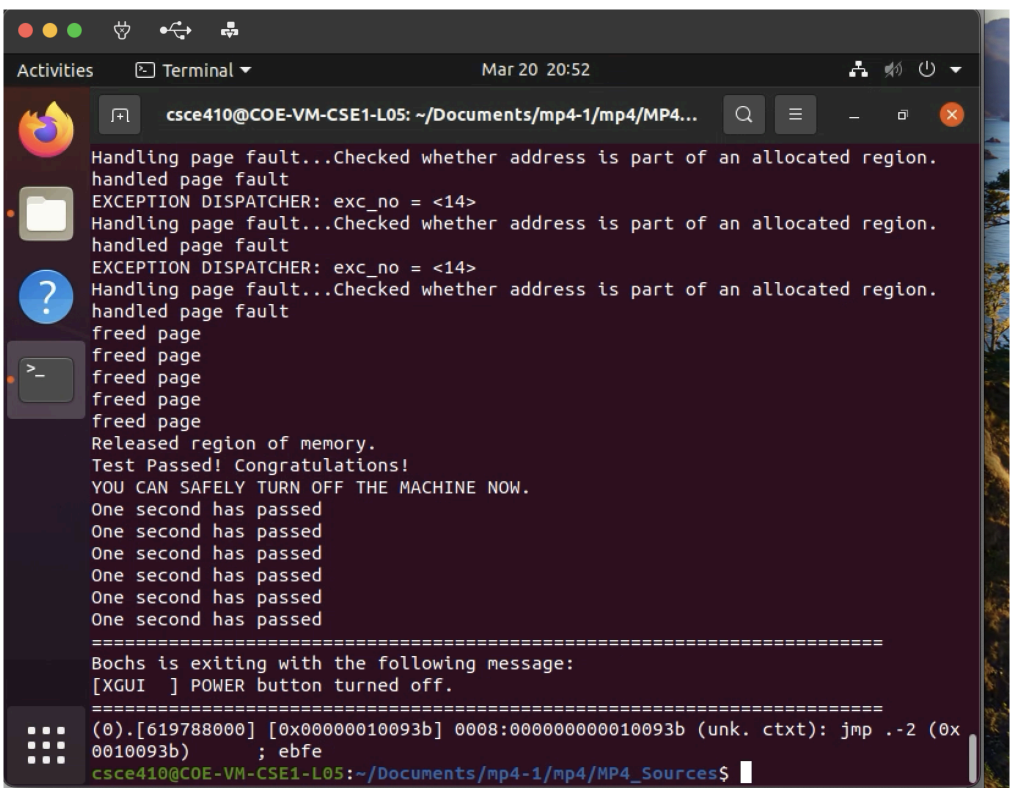


Testing the recursive page table implementation



Testing the vm pool implementation


Activities
Terminal
Mar 20 20:28
csce410@COE-VM-CSE1-L05: ~/Documents/mp4-1/mp4/MP4...
Installed exception handler at ISR <0>
Installed interrupt handler at IRQ <0>
Installed interrupt handler at IRQ <1>
after installing keyboard handler
Frame Pool initialized
Frame Pool initialized
Installed exception handler at ISR <14>
Initialized Paging System
Constructed Page Table object
Loaded page table
Enabled paging
Hello World!
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...handled page fault
EXCEPTION DISPATCHER: exc_no = <14>


Activities
Terminal
Mar 20 20:52
csce410@COE-VM-CSE1-L05: ~/Documents/mp4-1/mp4/MP4...
Handling page fault...Checked whether address is part of an allocated region.
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...Checked whether address is part of an allocated region.
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
Handling page fault...Checked whether address is part of an allocated region.
handled page fault
freed page
freed page
freed page
freed page
freed page
Released region of memory.
Test Passed! Congratulations!
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
==
Bochs is exiting with the following message:
[XGUI   ] POWER button turned off.
==
(0).[619788000] [0x00000010093b] 0008:000000000010093b (unk. ctxt): jmp .-2 (0x0010093b)   ; ebfe
csce410@COE-VM-CSE1-L05:~/Documents/mp4-1/mp4/MP4_Sources$