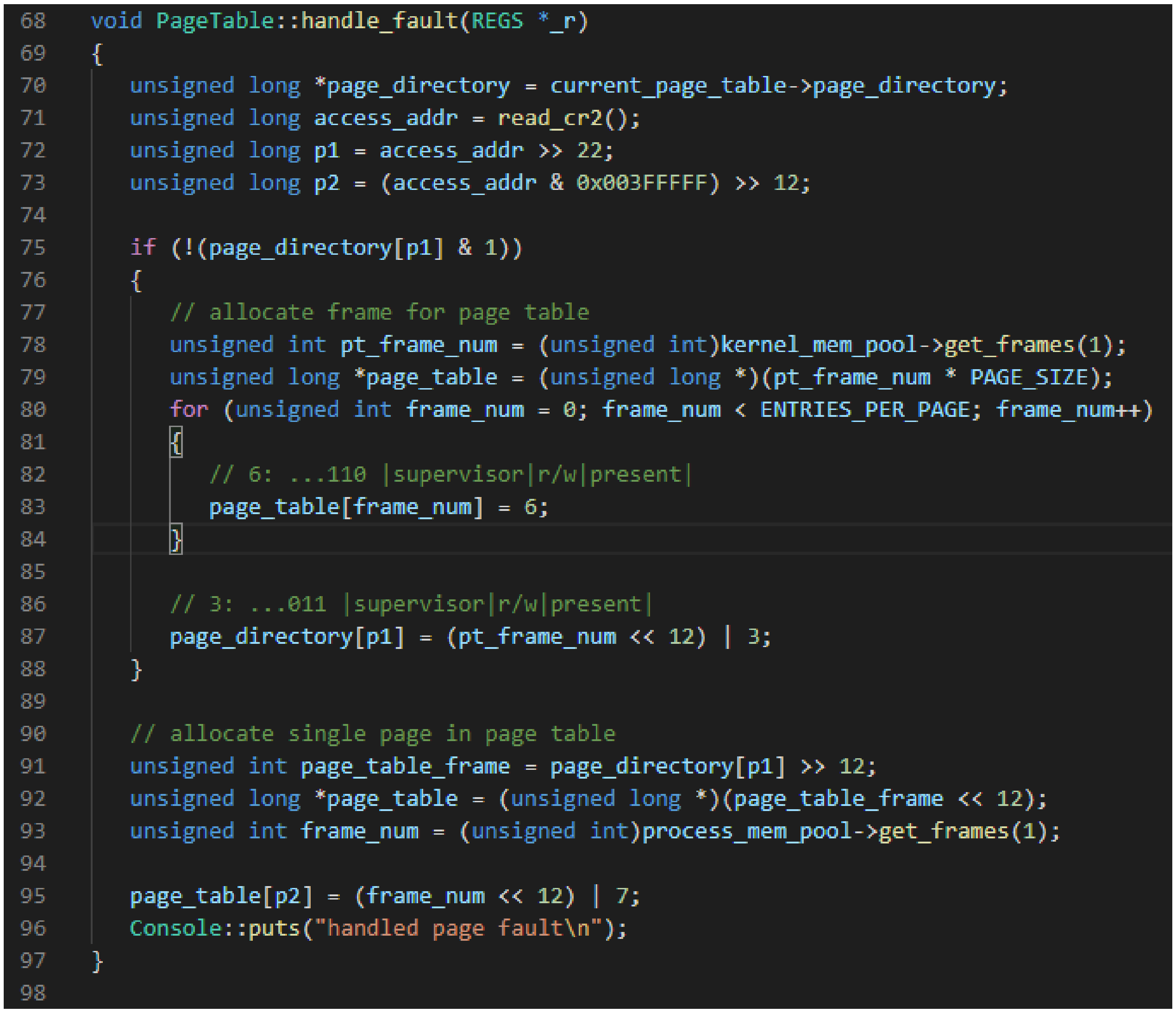
The page table and page directory will need to be allocated in process memory. This means that all places where I get frames from “kernel_mem_pool” will need to be changed to “process_mem_pool”: in PageTable constructor with lines 26, 30, and in PageTable “handle_fault” function with lines 78.

The recursive strategy requires that the last entry in the page directory points to the page directory itself. This means that in the constructor when we create the page directory we will set the last entry “page_directory[1023] = page_directory | 1” where 1 means that the entry is supervisor, read access, and present. The for loop on line 44 in the PageTable constructor will need to also iterate to “page_table_num < ENTRIES_PER_PAGE - 1”, so it does not mark the last frame in the page directory.

When handling page fault, it is important to remember the addressing is in virtual memory. There are a few key portions that need to be changed to accommodate this. When we get the page directory address, it should be converted to “|1023|1023|0|” on line 70. This means that whenever we index the page directory for the page table, say “page_directory[p1]” on line 75, this address gets converted to “|1023|1023|p1,00|”. This is the ideal logical address to get the p1th page table because the recursive strategy will grab the page directory twice and then the p1th page table. On line 79 where we get the page table address, we need to change this to be “|1023|p1|0|”, so the for loop on line 80 will index values properly “page_table[frame_num]” will be logical addresses “|1023|p1|frame_num,00|”. The next change is on line 87: we need to set the availability of the page table before we start the for loop on line 80. This prevents more page faults because those entries would not be accessible in the page table that is marked not present.

Code for reference:



```
void PageTable::handle_fault(REGS *_r)
{
    unsigned long *page_directory = current_page_table->page_directory;
    unsigned long access_addr = read_cr2();
    unsigned long p1 = access_addr >> 22;
    unsigned long p2 = (access_addr & 0x003FFFFF) >> 12;

    if (!(page_directory[p1] & 1))
    {
        // allocate frame for page table
        unsigned int pt_frame_num = (unsigned int)kernel_mem_pool->get_frames(1);
        unsigned long *page_table = (unsigned long *)(pt_frame_num * PAGE_SIZE);
        for (unsigned int frame_num = 0; frame_num < ENTRIES_PER_PAGE; frame_num++)
        {
            // 6: ...110 |supervisor|r/w|present|
            page_table[frame_num] = 6;
        }

        // 3: ...011 |supervisor|r/w|present|
        page_directory[p1] = (pt_frame_num << 12) | 3;
    }

    // allocate single page in page table
    unsigned int page_table_frame = page_directory[p1] >> 12;
    unsigned long *page_table = (unsigned long *)(page_table_frame << 12);
    unsigned int frame_num = (unsigned int)process_mem_pool->get_frames(1);

    page_table[p2] = (frame_num << 12) | 7;
    Console::puts("handled page fault\n");
}
```