Monami Dutta Gupta

OS Assignment 2 – Part 2

3.1

For this part, we need to define a TSS segment.
Firstly, I have added two zero segments to my kernel_entry.S. like this → .quad 0x0000000000000000.
These are the two entries for a TSS descripter which will be initialized in load_tss_segment() function in interrupts.h.

In boot.c:
I’m allocating two extra pages: 1 for rsp[0] stack and the other for tss.

In interrupts.h:
IN the init_tss_segment() function, I’ve initialized the tss_segment struct by assigning the required values – mainly the rsp stack address to rsp[0] and size of the structure to iopb_base.
The ist fields and other fields have been set to 0.

In the load_tss_segment() function, I’m passing the selector as 0x28 corresponding to the 5th entry in gdt. (8*5 = 40, which is 0x28)

I’m assigning the base as the tss address (same as the rsp[0] stack bottom address) and the limit as the size of the structure – 1.

This initializes and loads the tss segment in the code.

3.2

Firstly, I created the idt_descriptor structure as an array of size 256 and 16-byte aligned.

In kernel.c:
The x86_init() function is calling the init_idt_table() function which I’m using to initialize the idt descriptor structure by setting dpl as 0, p as 1, type as 14, ist as 0 and selector as 0x8.

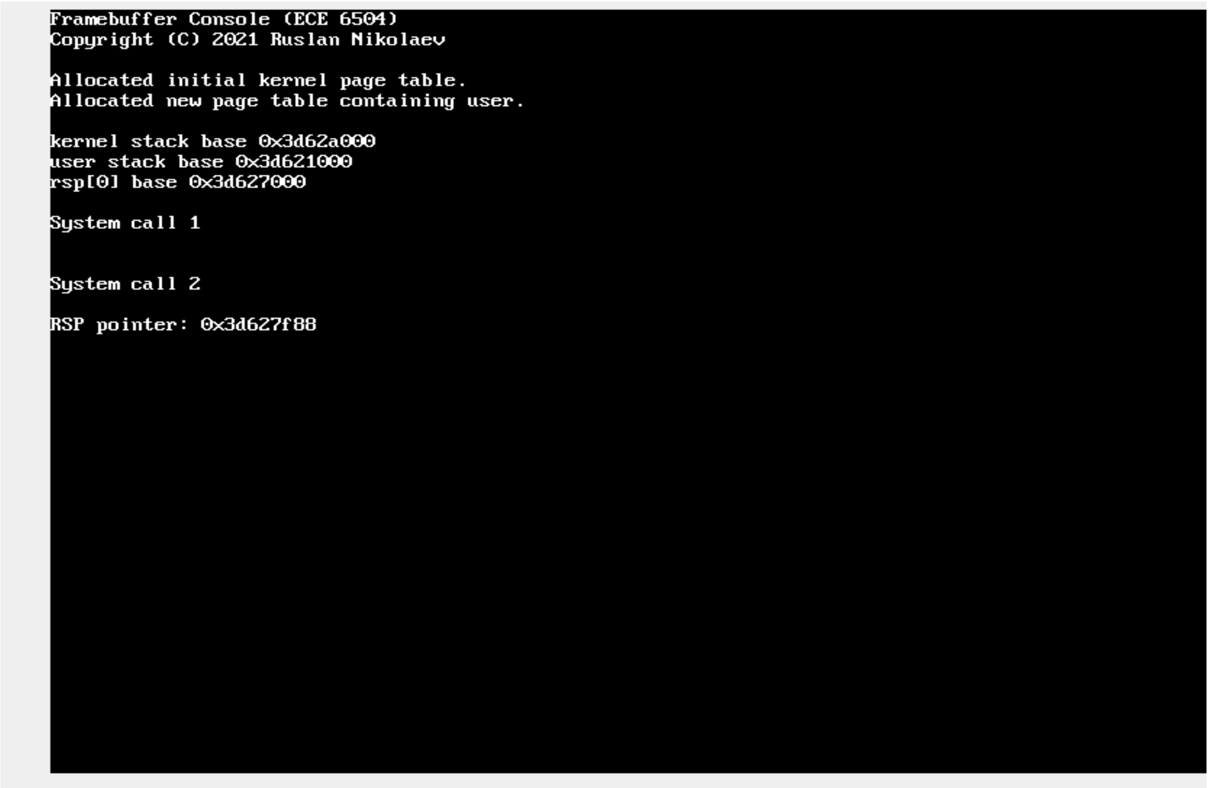
The offset values of the first 32 entries come from the default trap pointer address, and the rest are assigned 0.

I’m initializing the idt_pointer by setting the size of the idt table -1 and base as the address of the idt table.

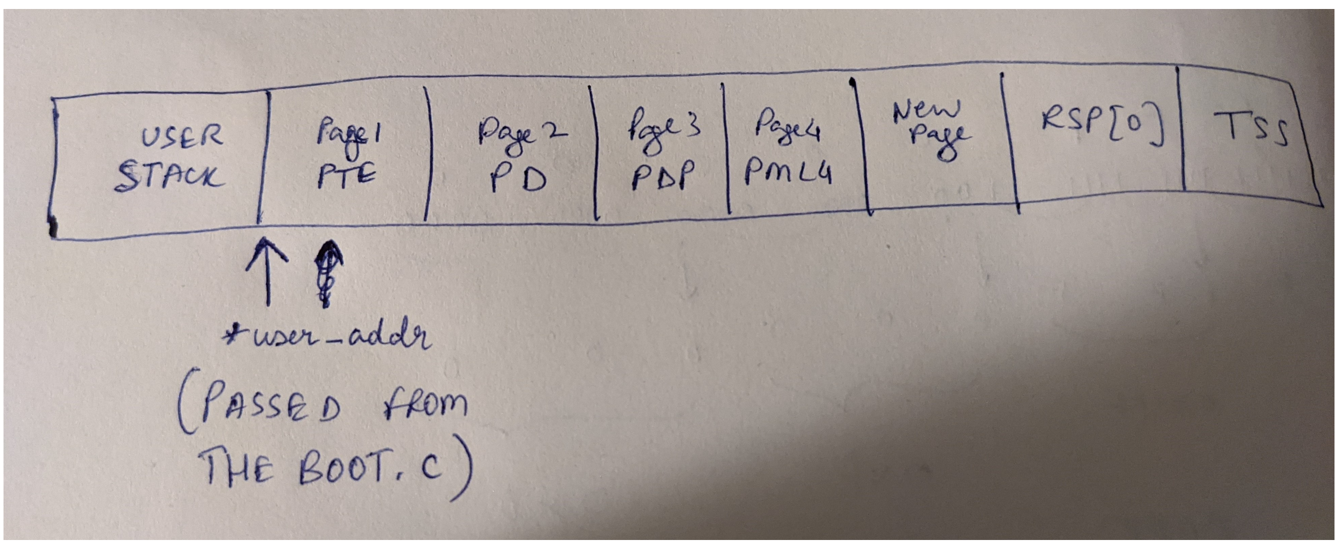
In kernel_entry.S, I’m assigning two pointers to point to the default and page fault handler respectively.

In kernel_asm.S,
under default_trap, I’m calling the default_handler() function and passing the rsp value as an argument using the rdi register.

The default_handler is defined in kernel.c and it prints out the rsp pointer value. I try to access a non-existent address, which triggers the default handler. As we can see in the screenshot below, the rsp value is bigger than the rsp[0] stack base address and lesser than +0x1000, which means the stack is occupied.



Reference:
Picture of my user_addr buffer which might help in getting a clearer picture of the pointer arithmetic I have done in pagefault_handler to access the pte and pml4 addresses from the rsp[0] stack address.



3.3

Here we allocate the 15th IDT entry which executes page faults to a different handler called the page fault handler.

In boot.c,
I allocate 1 extra page to set up the last entry in my User PTE to handle the page fault.

In kernel.c,
While defining the initial user PTE I set up the last entry (511 entry) with address 0 and present 0, but with write as 1 and user bit as 1 (basically, 0x6).

I've already set up the page fault pointer in kernel_entry.S to point to my kernel handler function.

In kernel_asm.S,
under pagefault_trap, I'm calling the pagefault_handler() function.
This handler which is defined in kernel.c, accesses the pte[511] entry and puts the address as the newly retrieved address of the page (lazy load page) from boot.c and sets present as 1.

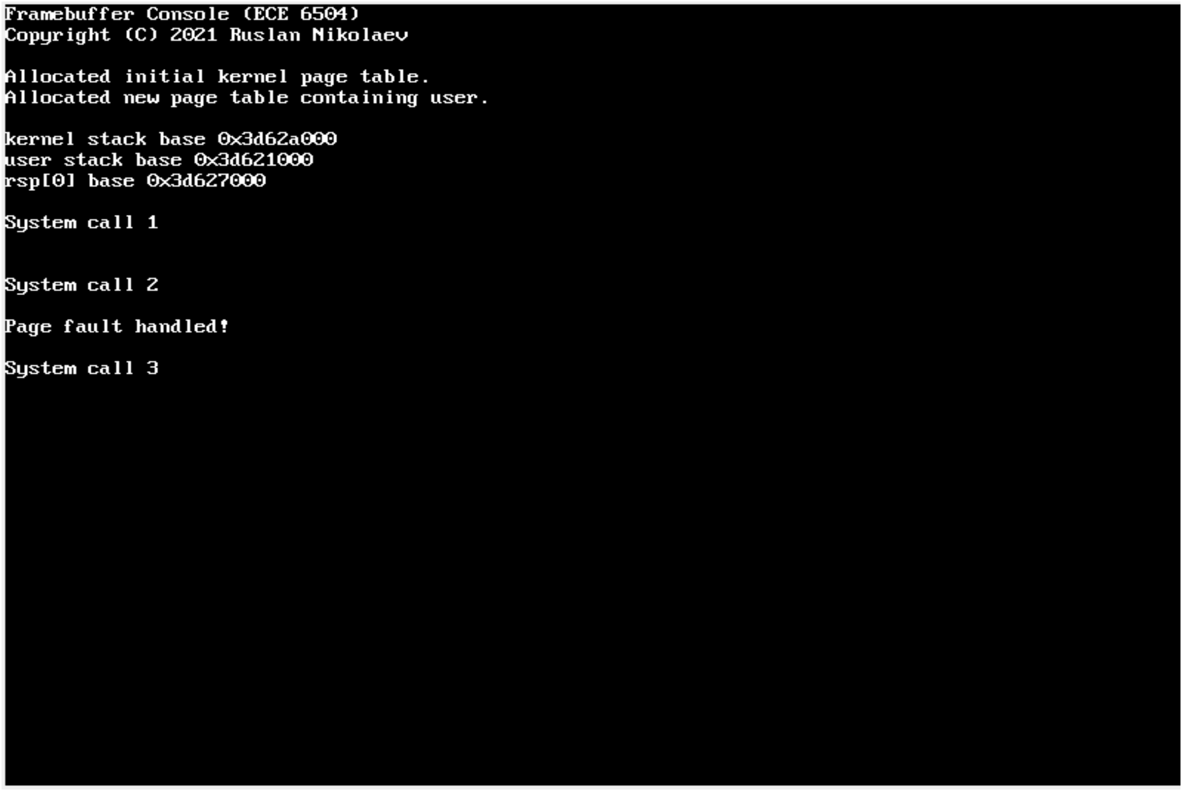
Then, I reload the cr3 register.

So, now when I try to access the non-existent address, my page fault handler gets executed and that address is not non-existent anymore.
I'm printing "page fault handled!" message as the last statement in my handler function.

This statement gets executed in my code right after I reload CR3, which goes to show that my page table has been correctly set up. I verify that the last entry in my page table contains the address of the lazy load page.

I've kept rsp_base_addr as the global variable which contains the rsp[0] base address, and I'm calculating pte and pml4 values based on that, since they are from the same buffer.

Then I'm executing a system call which prints the message "System call 3!" which shows the program is still alive.


Framebuffer Console (ECE 6504)
Copyright (C) 2021 Ruslan Nikolaev
Allocated initial kernel page table.
Allocated new page table containing user.
kernel stack base 0x3d62a000
user stack base 0x3d621000
rsp[0] base 0x3d627000
System call 1
System call 2
Page fault handled!
System call 3