Phase 2 Detailed Documentation: Memory Page Table and Long Mode

Mohamed Ayman Mohamed 900182267

Nada El-Shenawy 900182782

Abderlhman Shabaan 900183004

**Assumption taken:**

We did not assume anything other than the given instructions from the phase itself.

**Findings:**

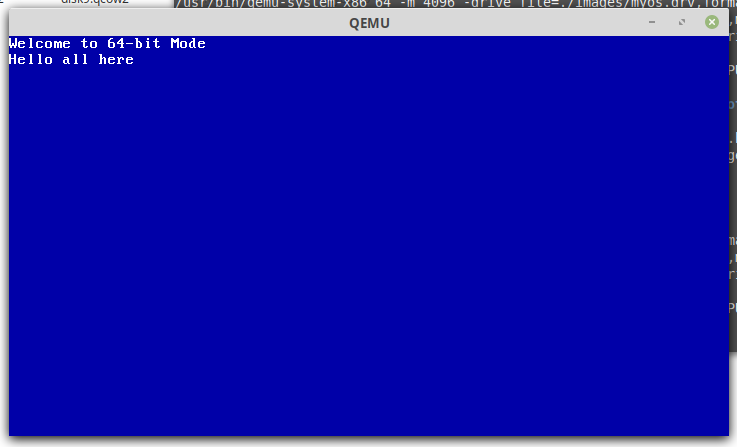
**Explanation to what we have done:**

In **the second stage boot loader**, we set up the Global Descriptor Table (GDT), by setting up the null, code and data descriptors. We then built a page table to map the first 2 MB of memory, which is needed to be able to switch to long mode. That was done by initializing 4 memory pages (1 for each level, 4kb each). We looped over the 2 MB of memory and mapped them as 4kb pages. We then configured the control registers CR4 (enabling bits 5&7) and CR3 (storing address of PML4), the special register EFER MSR (enabling long mode bit 8) and control register CR0 (to enable protected mode and paging) and then finally we jump to code segment in 64-bit mode. We have also added the part of video\_clear. We have made the screen to be blue using the part 20 of 0x1020. The most important part in the video\_clear We can use also 0x2020 for make it green and so on. It depends on which color you need to clear the screen in 32-bit mode. Now, we can move to the third stage.

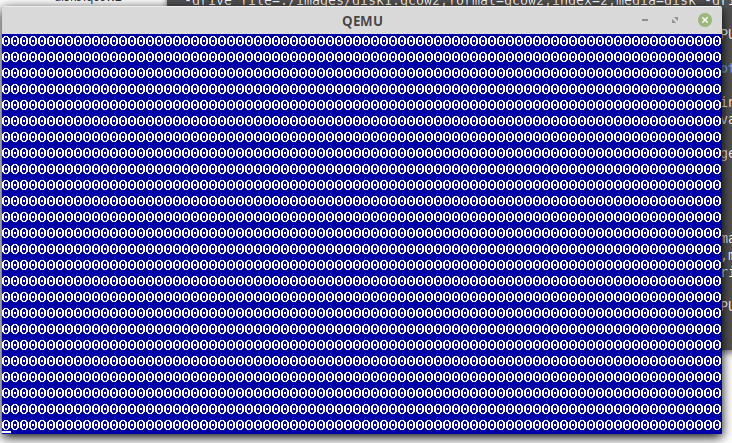
To be able to use unlimited memory and 64-bit registers, we need to **construct a full page table that can map the whole memory.** In this stage, we have constructed three files. Let's take it one by one. **First,** we need to map the first Two\_MB of the memory to make everything in state and work fine. This can be made by constructing 3 levels of page tables: "Each one is 4KB (512 entries \* 8 Bytes)". Now, PML4 (first Page Table) has been created: we can create the first Entry of PML4. We will check whether the PDP is created or not. If not, we will create it. After then, this entry will point to the next level page table, which is PDP. We will, then, check whether the PDT is created or not. If not, create it. In this step, PDP first entry contains the address of PDT. Finally, the PDT will contain the address of first two MB (0x000 in QEMU). At this point, we need to create all memory. Before explaining further, it is highly needed to explain the bitmap. To make it easier to ourselves, we have created a very small chunk of memory containing some important information to be able to construct the page table. Imagine this chunk as a vector of vector. In other words, I will have n segments of memory. Each segment contains the 8 bytes for first address, 8 bytes for length of this segment, a bit for checking whether this segment is 2MB(with 1) or 4KB(with 0), and the final bit indicating for whether it is mapped or still free. **Back to Page\_Walk,** we have a very small loop that calls map function to construct an entry. I will map the whole 2MBs segments first by checking the 17th bit of the segment in each segment, right? Then, we will do the same as what we have done with the first 2MB but with a little caring that we add on virtual address 2MB. With 4KB, we do the same by getting the segments of 4KB but here we add 4KB and map in 4th level (PTE).

**Some Screenshots for the running code:**

**With Write only:**



With read and write.



How to run the code:

* Just write: make run\_myos.