Assignment 4

Please complete the report and submit `report.pdf` through Blackboard

1. Q1 [20 pts]

Read Chapter 15 of "Three Easy Pieces" (https://pages.cs.wisc.edu/~remzi/OSTEP/vm - mechanism.pdf) and explain how do the CPU hardware and the operating system cooperate in the procedure of address translation.

2. Q2 [20 pts]

Read Chapter 16 "(https://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf) and chapter 18 (https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf) of "Three Easy Pieces" and compare segmentation and paging. Your answer should cover all aspects (e.g., size of chunks, management of free space, context switch overhead, fragmentation, status bits and protection bits, etc.) and compare them side-by-side.

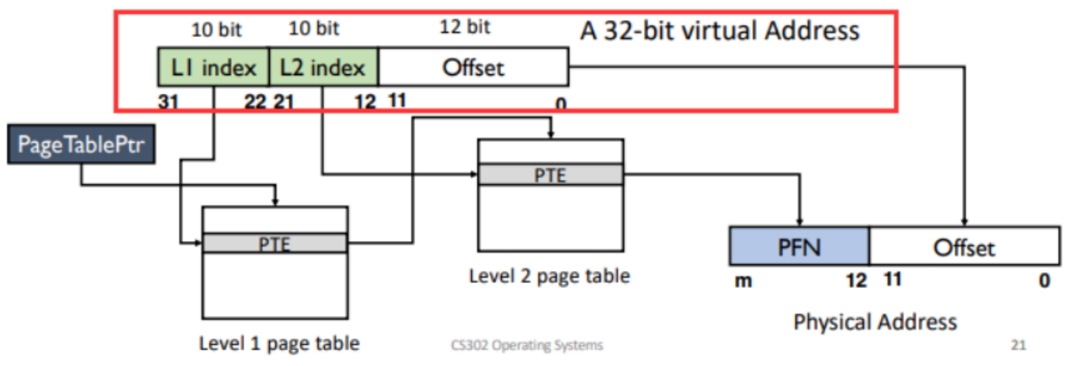
3. Q3 [20 pts]

Consider a system with the following specifications:

- 46-bit virtual address space
- Page size of 8 KBytes
- Page table entry size of 4 Bytes
- Every page table is required to fit into a single page

How many levels of page tables would be required to map the entire virtual address space? Please document the format of a virtual address under this translation scheme. Briefly explain your rationale.

> Hint: Here is the example of the format of a 32-bit virtual address in lecture.

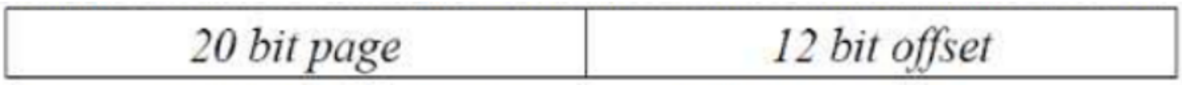


4. Q4 [20 pts]

Consider a system with following specifications:

- Both virtual address space and physical address are 32bits.
- Page table entry size of 4Bytes

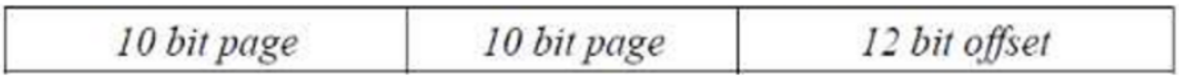
(a) Suppose it uses 1-level page table, the format of the translation scheme is



20 bit page	12 bit offset

What is the page size? What is the maximum page table size?

(b) Suppose it uses 2-level page table, the format of the translation scheme is



10 bit page	10 bit page	12 bit offset

- Please write down the 1-st level page number and its offset in decimal(base 10) of virtual address **0xC302C302** (base 16).
- Please write down the 2-nd level page number and its offset in decimal(base 10) of virtual address **0xEC6666AB** (base16)

5. Q5 [20 pts]

Please use the `buddy_system_allocator` crate to do allocation and deallocation, here are the recommended initialization, and corresponding `alloc` and `dealloc` functions:

```
let mut allocator: buddy_system_allocator::FrameAllocator<32> =
buddy_system_allocator::FrameAllocator::new();
allocator.add_frame(0, 64);

let base = allocator.alloc(4).unwrap();
allocator.dealloc(base, 4);
```

Steps:

1. allocate 4 frames
2. allocate 4 frames
3. deallocate the 4 frames in step 2
4. allocate 8 frames
5. allocate 2 frames

Output the starting and ending addresses of the frames allocated in steps 1, 2, 4, and 5, and take a screenshot. Please provide detailed explanations for each allocation.

Output template:

```
[Step 1 allocation] start: {}, end: {}.
[Step 2 allocation] start: {}, end: {}.
[Step 4 allocation] start: {}, end: {}.
[Step 5 allocation] start: {}, end: {}.
```