ID: 5140370941
Name: Jiacheng Ma
Exercise for OS-Lab2

Question 1. Assuming that the following JOS kernel code is correct, what type should variable x have, uintptr_t or physaddr_t?

- uintptr_t, after booting and setting up page table, JOS use virtual address rather than physical address.

Question 2. What entries (rows) in the page directory have been filled in at this point? What addresses do they map and where do they point? In other words, fill out this table as much as possible:

Entry	Base Virtual Address	Points to (logically):
1023	0xffc00000	Page table for top 4MB of phys memory
1022	0xff800000	?
960	0xf0000000	Remapped physical memory
958	0xef800000	Kernel stack (the upper half)
957	0xef400000	Current page table (kern_pgdir)
956	0xef000000	Table of information of all pages
1	0x00400000	
0	0x00000000	[see next question]

Question 3. (From Lecture 3) We have placed the kernel and user environment in the same address space. Why will user programs not be able to read or write the kernel's memory? What specific mechanisms protect the kernel memory?

- In the page table, the entry of kernel memory space are set as supervisor (PTE_U) and thus when user programs try to read or write it, MMU will find that the page is not allowed to be accessed by ring 3, which is the privilege level that user program executes on, so a page fault will happen to protect the page.

Question 4. What is the maximum amount of physical memory that this operating system can support? Why?

- 256 MB, because JOS will map the whole physical memory at 0xf0000000, and the largest memory address is 0xffffffff, and there's only 256MB between them.

Question 5. How much space overhead is there for managing memory, if we actually had the maximum amount of physical memory? How is this overhead broken down?

- 512KB to store the array of struct Page and 260KB to store the page table that makes it possible to access all pages.
- In some occasion, 4M large pages are used to reduce the number of page table.

Question 6. Revisit the page table setup in kern/entry.S and kern/entrypgdir.c. Immediately after we turn on paging, EIP is still a low number (a little over 1MB). At what point do we transition to running at an EIP above KERNBASE? What makes it possible for us to continue executing at a low EIP between when we enable paging and when we begin running at an EIP above KERNBASE? Why is this transition necessary?

- When jmp *%eax is executed, EIP is set to relocated, which is above KERNBASE.
- At that time, the low physical memory is mapped to low virtual memory and virtual address above KERNBASE at the same time. When visiting these two address, actually we are visiting the same range of physical memory.
- Before we enable paging, EIP is at the low part of memory address, only if there's a transition part of code could the execution of code properly.