REPORT LAB 2

Questions :

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

```
mystery_t x;
char* value = return_a_pointer();
*value = 10;
x = (mystery_t) value;
```

Ans: Since value is a pointer, it is a virtual address in C. So, x should preferably be of type “uintptr_t”
(Because x is a virtual address at which 10 is stored).

2)
MAPPINGS PRESENT:

[KERNBASE, 2^32) → [0, 2^32 - KERNBASE)
[UPAGES, UPAGES + size_PAGES) → [PAGES, PAGES + size_PAGES)
[PAGES, PAGES + size_PAGES) → [PAGES, PAGES + size_PAGES)
[bootstack, bootstack + KSTKSIZE) → [KSTACKTOP-KSTKSIZE, KSTACKTOP)

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?

ANS:

User processes are allocated pages from free_pages_list. User can access pages only if the page has 'PTE_U' permission . So

1) Kernel Pages are already allocated and never in free_pages_list.

2) All the pages of Kernel **do not** have 'PTE_U' permission i.e. users can't access kernel pages due to permissions set in the page table entries.

4)

What is the maximum amount of physical memory that this operating system can

support? Why?

ANS:
The maximum amount of physical memory that JOS can support is 256 MB. Reason for this is that user space has to be mapped to kernel space which is from 0xf0000000 to 0xffffffff which accounts for 256 MB

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?

ANS:
We have
1) Pages structure → NPAGES*sizeof(struct PageInfo); → NPAGES ref. To no.of
    pages → 64*1024 Pages of 4KB size.
2) Page directory → 4KB
3) Page Tables and entries in it → 64*4 KB = 256KB
    Since page table has 1024 entries it can reference 1024 pages.
Now 256 MB / 4KB → 64*1024 Pages → 64 Page Tables → 64*4KB

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?

ANS:
We transition to KERNBASE by the jump to relocated which is in eax just before entering C code. We can still continue our execution because there is a mapping set up from [0,4MB) in virtual memory to [0,4MB) in physical memory. This transition is necessary because we would like to clear away the lower map and make it available to user processes keeping kernel in upper part.