Question 1:

mystery\_t should be uintptr\_t as it points to a virtual address. We know that it points to a virtual address because it is set to the value of the pointer "value" which is dereferenced in the third line. That would make no sense if it were a physical address, since all address

translation starts with virtual addresses.

Question 2:

Entry Number Base Address Points to (Logically)

0x3c1-0x3ff 0xf0400000 All of the rest of phys memory

0x3c0 0xf0000000 First 4 MB of physical memory

0x3be 0xef800000 Kernel stack

0x3bd 0xef400000 Cur. Page Table

0x3bc 0xef000000 RO PAGES

Question 3:

The permissions for the kernel memory is set to be readable/writable only by the kernel. User processes will only be able to access pages marked with PTE\_U. This prevents user processes from modifying or reading kernel memory. The "pages" array is exposed to user processes through a mapping that marks them as PTE\_U without write access. This allows them

to read that data, but not modify it.

Question 4:

The maximum is 256MiB of memory. This is because the kernel expects all of physical memory to be mapped from KERNBASE to 2^32. This means that only 2^32-KERNBASE memory can be mapped. Since KERNBASE=0xf0000000 then 0x100000000 - 0xf0000000 = 0x10000000 = 256\*(1024^2) = 256MiB.

Question 5:

The overhead can be broken down into space to hold the page tables for the mapping of all physical memory and space to hold the "pages" array. There are 67 page tables required to hold the mappings (from question 2) plus one page for the page directories. This means that 272KiB is the constant amount of overhead for the page tables. If we have the maximum of

256MiB = 65536 pages of memory, we need 8 bytes (the size of struct Page) for each page. The overhead in the pages array is therefore 512KiB. Total, the overhead is 784KiB.

Question 6:

We transition to running at an EIP above KERNBASE when we make the indirect jump on line 67 to the address stored in %eax. Due to how linking was set up, addresses within the kernel are linked above KERNBASE. So when mov $relocated, %eax is executed, $relocated refers to the address of that instruction in virtual memory above KERNBASE. We may run without

error at the lower EIP value because of the first page directory entry and the page directory entry for KERNBASE being mapped to the same physical memory at the bottom of memory. This makes the first 4MB of virtual memory identity mapped to the first 4MB of physical memory, and the first 4MB of memory above KERNBASE mapped to that same memory. There are no errors because no data stored in the kernel is accessed before virtual memory is set up. If any of that memory were to be accessed before that mapping was set up, then there would be an error. This kind of transition is necessary because we want the kernel to be above KERNBASE in virtual memory, but we probably do not have KERNBASE bytes of physical memory. Therefore, we must load the kernel to a lower address and then transition to the new address before continuing. We cannot keep executing at a low EIP because that virtual memory is going to be used by user processes.