```
boot_map_region(kern_pgdir, KERNBASE, 0xffffffff-KERNBASE, 0, PTE_W | PTE_P);
```

After boot_map for Metadata for Users and kernel.

Entry 957 : Kern Page Directory

Entry 956 : Upages

```
boot_map_region(kern_pgdir,  UPAGES,          sizeof(struct PageInfo) * npages, PADDR(pages), PTE_U | PTE_P);
boot_map_region(kern_pgdir, (uintptr_t) pages, sizeof(struct PageInfo) * npages, PADDR(pages), PTE_W | PTE_P);
```

```
Entry : 956, address :f0118ef0, PDE: 3fd007
Entry : 957, address :f0118ef4, PDE: 118005
Entry : 960, address :f0118f00, PDE: 3fe007
```

Entry 959: Stack

```
boot_map_region(kern_pgdir, KSTACKTOP - KSTKSIZE, KSTKSIZE, PADDR(bootstack), PTE_W | PTE_P);
```

```
Entry : 956, address :f0118ef0, PDE: 3fd007
Entry : 957, address :f0118ef4, PDE: 118005
Entry : 959, address :f0118efc, PDE: 3ff007
Entry : 960, address :f0118f00, PDE: 3fe007
```

Rest are for the rest of Kernel Data from 0xf0000000 to 0xffffffff

```
kern_pgdir[PDX(UVPT)] = PADDR(kern_pgdir) | PTE_U | PTE_P;
```

```
 we are at the end of check_page_free_list
check_page_installed_pgdir() succeeded!
Entry : 956, address :f0118ef0, PDE: 3fd007
Entry : 957, address :f0118ef4, PDE: 118005
Entry : 959, address :f0118efc, PDE: 3ff007
Entry : 960, address :f0118f00, PDE: 3fe027
Entry : 961, address :f0118f04, PDE: 3fc027
Entry : 962, address :f0118f08, PDE: 3fb027
Entry : 963, address :f0118f0c, PDE: 3fa027
Entry : 964, address :f0118f10, PDE: 3f9027
Entry : 965, address :f0118f14, PDE: 3f8027
Entry : 966, address :f0118f18, PDE: 3f7027
Entry : 967, address :f0118f1c, PDE: 3f6027
Entry : 968, address :f0118f20, PDE: 3f5027
Entry : 969, address :f0118f24, PDE: 3f4027
Entry : 970, address :f0118f28, PDE: 3f3027
Entry : 971, address :f0118f2c, PDE: 3f2027
Entry : 972, address :f0118f30, PDE: 3f1027
Entry : 973, address :f0118f34, PDE: 3f0027
Entry : 974, address :f0118f38, PDE: 3ef027
Entry : 975, address :f0118f3c, PDE: 3ee027
Entry : 976, address :f0118f40, PDE: 3ed027
Entry : 977, address :f0118f44, PDE: 3ec007
Entry : 978, address :f0118f48, PDE: 3eb007
Entry : 979, address :f0118f4c, PDE: 3ea007
Entry : 980, address :f0118f50, PDE: 3e9007
```

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?

Because we used the permission bits such as PTE_U bit to prevent users, so whenever this bit is disabled, the user cannot read/write. This is performed by the hardware.

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

The kernel maps 1 page for every PageInfo struct as each page stores the metadata. each PageInfo is composed of a 32-bit pointer and a 16-bit reference count.

So , it will cost 8 bytes to store.

Now in one page which is 4kb a total amount of 512 PageInfo can be stored in this page.

These page info describe each page The total amount of physical memory it can support is 2GiB.(512*4K)

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?

Now we know that the total amount of physical memory is 2GB = 2^31 . if we use all memory we can map 2^31/4kb(which is size of a page) = 2^19 pages . To index this we need 2^22 bytes so its 4MB(not sure of this question)

For the whole 4GB
Page info array ⇒ # pages * 8 =
Page directory ⇒ 4KB (just one page)
Page tables ⇒ 102l4 page tables = 1024 * 4 KB =

For 2 GB
# of pages `4MB/8B=512K` pages
Page info array ⇒ # pages * 8 = 4MB
Page directory ⇒ 4KB (just one page)
Page tables ⇒ 1024 page tables = 1024 * 4 KB =

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?

The paging is enabled after mov %eax, %cr0 instruction.

```
Breakpoint 1, 0x0010001a in ?? ()
(gdb) si
=> 0x10001d:    mov    %cr0,%eax
0x0010001d in ?? ()
(gdb) si
=> 0x100020:    or     $0x80010001,%eax
0x00100020 in ?? ()
(gdb) si
=> 0x100025:    mov    %eax,%cr0
0x00100025 in ?? ()
(gdb) si
=> 0x100028:    mov    $0xf010002f,%eax
0x00100028 in ?? ()
(gdb) si
=> 0x10002d:    jmp    *%eax
```

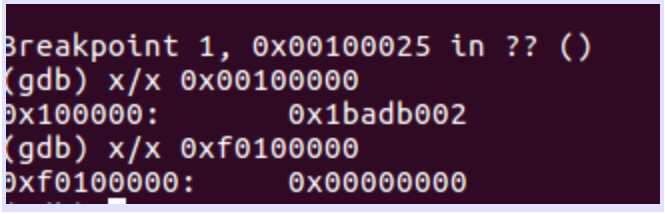
We can see in the above screenshot that even after enabling the paging, we are still using low EIP, we can do that because :-

```
__attribute__((__aligned__(PGSIZE)))
pde_t entry_pgdir[NPDENTRIES] = {
        // Map VA's [0, 4MB) to PA's [0, 4MB)
        [0]
                = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P,
        // Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
        [KERNBASE>>PDXSHIFT]
                = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P + PTE_W
};
```

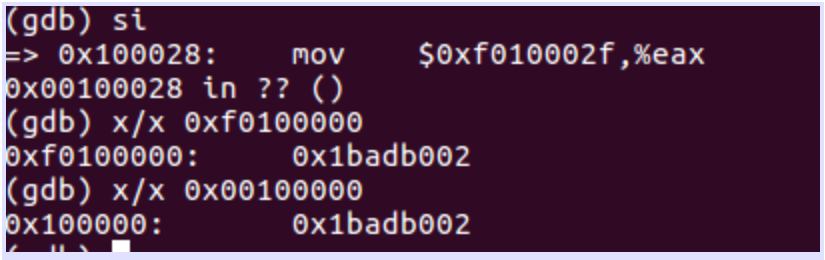
It jumps above KERNBASE after the jmp *%eax instruction as seen below:

```
=> 0x100028:    mov    $0xf010002f,%eax
0x00100028 in ?? ()
(gdb) si
=> 0x10002d:    jmp    *%eax
0x0010002d in ?? ()
(gdb) si
=> 0xf010002f:  mov    $0x0,%ebp
0xf010002f in ?? ()
(gdb)
```

Before the instruction movl %eax, %cr0 , there is no address translation



After the instruction, we can see that there is mapping/virtualization



It is necessary because after we finish initializing the page tables, `kern_pgdir will be loaded into cr3.`
`Also because the translation will only for 4 MB which is not enough to run the system.`