5137169011 陈伟业

Question：

1. uintptr_t

2.

Entry	Base Virtual Address	Points to (logically):
1023	0xffc00000	Page table for top 4MB of phys memory
1022	0xff800000	
959	0xefc00000	Kernel stack
957	0xef400000	Kern_pgdir
956	0xef000000	Pages array
2	0x00800000	
1	0x00400000	
0	0x00000000	[see next question]

3.用户程序如果访问并修改内核内存，有可能会导致内核崩溃。操作系统通过设置页表的 Supervisor/User 位为 0，使得用户不能访问这个页。

4. 4MB 存放页表信息，而每个 entry 为 8byte，故有 512K 个页。而页大为 4K，则内存大小为 4MB/8B*4KB=2GB

5. 整个 pageinfo 为 4MB，kern_pgdir 为 4KB，当前页表为 2MB，故总共 6148KB.

6. 在 entry.S 文件中有一个指令 jmp *%eax，将 EAX 的值赋给了 EIP，开始运行高于 KERNBASE 的代码。 页表将虚拟地址[0, 4MB)映射到了物理地址 [0, 4MB)，使得地址转换可以完成。

Implementation：

1.boot_alloc()：将空闲指针向后移动到相应的位置，返回一块空闲块。

2.page_init()：初始化 page_free_list，由高地址到低地址构建 list，其中去除 IO hole 到页表结束的区域。

3.page_alloc()：分配单个空闲块，当 alloc_flags & ALLOC_ZERO 为 TRUE 时，初始化页为全 0，如果 page_free_list 为空，则返回空。

4.page_free()：释放该页，将其按顺序找到位置插入 page_free_list。

5.pgdir_walk()：通过 va 和 pgdir 找到二级页表基地址，再使用 va 和 pde 计算出相应的物理地址，并返回对应的虚拟地址。

6. boot_map_region()： 通过 pgdir_walk()，找到 va 在页表中的位置，将对应的 pa 填入其中。

7. page_lookup()：通过 pgdir_walk()找到 va 对应的物理地址所在的虚拟地址，如果结果不为 NULL，则将其物理地址存放的数据转换为一个 page 返回。

8. page_insert()：先增加 pp->pp_ref，然后进行 remove，以防 free 空指针。

9. page_remove()：首先通过 page_lookup()找到相应的页，调用 page_decref 将其释放并减少 ref，然后调用 tlb_invalidate()，使得 tlb 中对应条目失效。

10. boot_map_region_large(): 在调用该函数先需在 cr4 开启 PTE_PS；对 64 个大页进行直接映射，即按传入的 va 和 pa 进行线性映射。

Challenge：

Showmappings：

Showmappings [begin] [end]

使用 pgdir_walk 遍历从 begin 到 end 的地址的页面，若 PTE_P 为 TRUE，则打印信息，否则输出该页为空。

Setmapping：

Setmapping [addr] [1|0] [P|U|W]

使用 pgdir_walk **找到** addr 所在页面，将其 P|U|W 位设为传入的参数（1 or 0）。

dumpMem:

dumpmem [p|v] [begin] [end]

通过 v 或 p 指定是虚拟地址还是物理地址，打印从地址 begin 到 end 的内存数据。

当需要访问虚拟地址 va 时，我将其指向的数据直接输出。

当需要访问物理地址 pa 时，我将 KERNBASE+pa 作为某个虚拟地址，输出其地址指向的数据。