Lab2 Solution

516030910006 方俊杰

Exercise 1.

完成 Physical Page Management，进行了内存信息获取、内存 PageInfo 数组初始化，同时初始化 `page_free_list` 保存空闲页链表，提供 `page_alloc()` `page_free()` 函数

Exercise 2.

阅读 Manual 关于 page translation 和 page-based protection 的内容。了解 Virtual, Linear, and Physical Addresses 三者的关系，在此可忽略 segmentation translation，认为前两种 addr 相同。
一旦进入 protected mode，所有的内存访问都使用的是 virtual address，不能再直接使用 physical address。

Eercise 3.

使用 `make qemu-nox` 打开 QEMU，不能用 `make qemu`，然后在 K> 使用 `ctrl + a c` 会进入内置 QEMU monitor 状态，注意 按 c 时松开 ctrl。
此时可以使用 `info mem`、`info registers`等查看内存和寄存器，用 `xp` 查看物理地址，与 GDB 类似。

Question

1. `x` 的类型是 `uintptr_t`，C中所有的指针值都是 virtual address，即前文对应的 uintptr_t。

Exercise 4.

实现有关 `page table` 管理的函数，包括通过页表映射的`pgdir_walk`，页表增删查的函数，每个函数通过获取 va 和 对应的 PTE，进行相应操作。

Exercise 5.

调用 `boot_map_region` 完成对几段虚拟地址的映射

Question

2.

Entry	Base Virtual Address	Points to (logically)
1023	0xffc0 0000	Page table for top 4MB of phys memory
1022	0xff80 0000	
960	0xf000 0000	KERNBASE
959	0xefff 8000	bootstack
956	0xef00 0000	UPAGES

Entry	Base Virtual Address	Points to (logically)
2	0x0080 0000	user programs generally begin
1	0x0040 0000	temporary page mappings
0	0x0000 0000	

3. 每块内存 kernel 和 user 都有其读写权限控制，对于 kernel 环境的内存，其权限位 `PTE_U` 不置位则用户不能读取，在 MMU 硬件处理内存映射时会进行检查

4. 2G，UPAGES 最大为 4MB，可以存放大小为 8 bytes 的 PageInfo 共计 512K 个，故对应 512K 个 page 即 2 G

5. 按上述计算 PDE: 4K, PTE: 2M, pages: 4M 共计 6M + 4K
   可以使用 large page 来减少 overhead

6. 在 `kern/entry.S` 中的 `jmp *%eax` 实现跳转，修改了 eip，设置为 KERNBASE 以上的值。
   因为 `entry_pgdir` 手动映射了 [0,4M] 的内存，使得在跳转前也可以运行。
   因为之后就要映射更多的虚拟地址了，[0,4M]的虚拟地址不可访问

Challenge

设置 large page，类似 `boot_map_region` 实现 `boot_map_region_large`，用 PDE 直接指向物理页，注意将 `PTE_PS` 置位。同时需要在初始化修改 `cr4` 开启大页映射位 `CR4_PSE`

`showmapping` 命令已添加，输入两个虚拟地址以输出在此范围内的内存权限状态，需要更多检查