Lab2 Document（include the challenge!）

姓名：程浩 学号：515080910012

The Implementation of Part I & II & III

通过注释，可以很好地了解到每一个函数需要我们添加的代码是什么，然后根据提示开始写。

- Part1
  - boot_alloc()：这个函数此时是暂时来当作页分配器，后续lab会使用真实的页分配器page_alloc()。这个函数主要作用就是维护一个static的nextfree，即存放的是下一个空闲空间的虚拟地址。那么每次分配内存后，都需要修改它。
  - mem_init()：第一处是让我们分配一块内存，用来存放PageInfo的数组，操作系统可以通过这个数组来追踪内存使用情况。
  - page_init()：这个函数包括初始化pages数组、初始化pages_free_list链表，后者存放着空闲页的信息。此处的循环会遍历所有PageInfo，并根据这个页的状态来修改这个struct的状态，若该页被占用，则PageInfo.pp_ref置为1，空闲的话则放入pages_free_list中。而注释中又提到了由三部分已经被占用。
  - page_alloc()：根据注释知道，该函数主要需要从空闲链表中取出一个空闲的PageInfo，然后修改空闲链表信息，同时修改该取出的PageInfo的信息，并进行初始化。
  - page_free()：修改对应PageInfo的信息并把它插入到pages_free_list中。
- Part2
  - pgdir_walk()：首先通过pgdir求得这个VA所在的页表页，并得到与之对应的页目录地址dic_entry_ptr；判断该页表页是否在内存中，若在，计算它的page_base，返回页表项地址&page_base[PTX(va)]，不在的话，同时如果create为true，那就分配新的页，并将之添加到dic_entry_ptr中，create为false则直接return null。
  - boot_map_region()：这个函数为了将[va, va+size]映射到[pa, pa+size]上，所以，通过循环，在每一轮将一个虚拟页和物理页的映射关系存放到相应dic_entry_ptr中。
  - boot_map_region_large()：根据注释，将上一函数的PGSIZE换为了PTSIZE，并增加了一个PTE_PS位。
  - page_lookup()：调用pgdir_walk()获取这个va对应页表项，判断它是否在内存里，在的话就返回PageInfo指针，并将该页表项存到pte_store里。
  - page_remove()：将va与物理页的映射关系移除，则，将这个页对应页表项置为0，pp_ref-1，如果为0则将它回收。
  - page_insert()：使用pgdir_walk()找出va对应页表项，修改其pp_ref，如果这个va已经被映射，则移除这个映射，然后将va与这个物理页的映射加入到页表项中。
- Part3
  - 完善mem_init()：首先将pages数组映射到线性地址UPAGES，大小为PTSIZE；然后根据注释提示将下一个划为两部分，即将[KSTACKTOP-KTSIZE, KSTACKTOP)这里的映射关系加入到页表中；最后一部分是映射整个内核，用到我们之前完成的boot_map_region_large()，但是要设置CR4来开启页扩展。

Challenge！（选择做的challenge2）

- 实现showmap()：通过使用strtol()将输入的地址作为十六进制存储，然后利用前面实现的pgdir_walk()得到该va对应页表项，拿到它的PTE值，并输出，用法：

  ```
  cprintf("Usage: showmappings 0x....(begin_address) 0x....(end_address)\n")
  ```

- 实现setpermission()：使用一样的方法存储要修改虚拟地址的十六进制，用法：

  ```
  cprintf("Usage: setpermission 0x...(address) ..(P or W or U) ..(0 or 1)\n")
  ```

- 实现dump()：对于逻辑地址来说，由于程序里已经是了，可以直接取值；而物理地址采用的是暴力的方法，使用KADDR转换为va再取值。用法：

  ```
  cprintf("Usage: dump ..(p or v) 0x...(begin_address) 0x...(end_address)\n")
  ```