Jos Lab 2: Memory Management

Li Xinyu 515030910292

Part 1: Physical Page Management

> **Exercise 1**
> `boot_init()` 只在系统启动时被用来分配内存，其中静态变量 `static char *next_free` 记录下一个可以使用的空闲页的起始地址。该函数分配n字节的空间并按照 `PGSIZE` 对齐，因此将 `next_free` 增加n后调用 `ROUNDUP(nextfree, PGSIZE)` 进行对齐，如果 `next_free` 所指的地址超过可用的物理地址空间则调用 `panic` ，否则返回分配空间的起始地址。
> `page_init()` 按照注释中所提到的四种情况，将第一个页(索引为0)、IO hole中的页、初始化时分配的页(在extend memory中)的pp_ref设置为1，表示已被使用。然后将剩下的页面的pp_ref设置为0并加到 `page_free_list` 中。
> `page_alloc()` 首先检查 `free_page_list` ，若为空则返回 `NULL` ，否则取其中的第一个空闲页，根据 `alloc_flags & ALLOC_ZERO` 是否为真来决定是否将页中所有字节初始化为0。
> `page_free()` 首先检查该页的 `pp_ref` ，若不为0说明被使用则调用 `panic` ，否则将其添加到 `page_free_list` 的头部。

Part 2: Virtual Memory

Page Table Management

> **Exercise 4**
> `pgdir_walk()` 首先分解出 `va` 中的 `pdx` 、 `ptx` ，然后根据 `pdx` 找到 `pgdir` 中对应的entry，如果该entry的 `PTE_P` 位不为1(即页失效或不存在)时，若 `create` 不为true则返回 `NULL` ，否则调用 `page_alloc` 分配一个页（分配失败时亦返回 `NULL` ）。然后将该页的 `pp_ref` 加一，将页的物理地址和权限填写到之前的entry中。此时entry的 `PTE_P` 位一定为1，然后根据entry获取相应page table的基地址，加上 `ptx` 即为参数 `va` 的页表项指针。
> `boot_map_region()` 将虚拟地址[va, va+size)映射到[pa, pa+size]，因此遍历地址空间[va, va+size)(遍历间隔为 `PGSIZE`)调用 `pgdir_walk` 找到虚拟地址对应的页表项，将其设置为 `pa | perm | PTE_P` 。
> `page_lookup()` 首先调用 `pgdir_walk` 找到参数 `va` 的页表项，若为空或 `PTE_P` 位为0则返回 `NULL` ，否则根据
> 页表项中的物理地址找到对应的页，若参数 `pte_store` 不为空则将页表项指针赋给它，最后返回页的指针。
> `page_remove()` 调用 `page_lookup` 找到参数 `va` 对应页表项和页，如果页为空则返回，否则调用 `page_decref` 并将页表项清空为0，使相应的TLB项失效。

> `page_insert()` 首先调用 `pgdir_walk` 找到参数 `va` 的页表项，若为空则返回 `-E_NO_MEM` 。如果该页表项的 `PTE_P` 位为1(说明已经被映射)，则将该页的 `pp_ref` 加一，调用 `page_remove` 删除先前映射的页并调用 `tlb_invalidate` 使该页表项的TLB失效。最后将页的物理地址和权限填写到页表项中。

Part 3: Kernel Address Space

Initializing the Kernel Address Space

> **Exercise 5**
> 根据 `mem_init` 中的注释调用 `boot_map_region` 完成虚拟地址到物理地址的映射如下：
> map ‘pages’ read-only by user at UPAGES :
> `boot_map_region(kern_pgdir, UPAGES, size, PADDR(pages), PTE_U);`
> map phys addr of ‘bootstack’ as kernel stack:
> `boot_map_region(kern_pgdir, KSTACKTOP - KSTKSIZE, KSTKSIZE, PADDR(bootstack), PTE_W);`
> map all of physical memory at KERNBASE:
> `boot_map_region(kern_pgdir, KERNBASE, ksize, 0, PTE_W);`

-

> **Exercise 6**
> `boot_map_region_large` 与 `boot_map_region` 类似，只不过将遍历间隔设置为 `PTSIZE` 并将页表项的权限加上 `PTE_PS` 。
> 然后将 `mem_init` 中设置KERNBASE映射的函数改为 `boot_map_region_large(kern_pgdir, KERNBASE, ksize, 0, PTE_W);` 并在加载 `CR3` 之前开启 `Page Size Extension` 即在 `CR4` 中加上权限 `CR4_PSE` 。

Challenge1:

I have no idea how generalize the kernel’s memory allocation system to support chunk allocation, maybe its’ a method to find a physically contiguous chunk by traversing the free list.

Challenge2:

- For displaying physical page mappings in a useful and easy-to-read format:
  添加函数 `showmappings` ，传入两个参数–起始地址和结束地址，然后遍历给定的地址区域调用 `page_walki` 获取对应的PTE，然后输出信息。示例如下：

```
K> showmappings 0xef01a000 0xef01f000
|  virtaddr  |  physaddr  | P | W | U | PWT | PCD | A | D | PS | G |
|-------------------------------------------------------------------|
| 0xef01a000 | 0x00134000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
| 0xef01b000 | 0x00135000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
| 0xef01c000 | 0x00136000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
| 0xef01d000 | 0x00137000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
| 0xef01e000 | 0x00138000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
| 0xef01f000 | 0x00139000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
K>
```

- For explicitly set, clear, or change the permissions of any mapping in the current address space:
  添加函数`mon_chperm`，传入至少三个参数–第一个参数表示地址，后面的参数表示要设置或者移除的权限，示例如下：

```
K> showmappings 0xef01a000 0xef01a000
|  virtaddr  |  physaddr  | P | W | U | PWT | PCD | A | D | PS | G |
|-------------------------------------------------------------------|
| 0xef01a000 | 0x00134000 | 1 | 0 | 1 |  0  |  0  | 0 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
K> chperm 0xef01a000 +W -U +A
K> showmappings 0xef01a000 0xef01a000
|  virtaddr  |  physaddr  | P | W | U | PWT | PCD | A | D | PS | G |
|-------------------------------------------------------------------|
| 0xef01a000 | 0x00134000 | 1 | 1 | 0 |  0  |  0  | 1 | 0 | 0  | 0 |
|-------------------------------------------------------------------|
K>
```

- For dumping the contents of a given virtual/physical address memory range:
  添加函数`mom_dumpcont`，传入三个参数–第一个参数表示传入的是虚拟地址还是物理地址，后面的两个为起始地址和结束地址，然后遍历给定地址区域，读取每个地址内所存的一个字节内容。示例如下：

```
K> dumpcont -v 0xf0100000 0xf0100010

0xf0100000: 02 b0 ad 1b
0xf0100004: 00 00 00 00
0xf0100008: fe 4f 52 e4
0xf010000c: 66 c7 05 72
0xf0100010: 04
K> dumpcont -p 0x100000 0x100010

0x100000: 02 b0 ad 1b
0x100004: 00 00 00 00
0x100008: fe 4f 52 e4
0x10000c: 66 c7 05 72
0x100010: 04
K>
```