Mitjos lab2 实验报告

李东闻 1511489

一、实验内容

Exercise 1. In the file `kern/pmap.c`, you must implement code for the following functions (probably in the order given).

`boot_alloc()`
`mem_init()` (only up to the call to `check_page_free_list(1)`)
`page_init()`
`page_alloc()`
`page_free()`

`check_page_free_list()` and `check_page_alloc()` test your physical page allocator. You should boot JOS and see whether `check_page_alloc()` reports success. Fix your code so that it passes. You may find it helpful to add your own `assert()`s to verify that your assumptions are correct.

```
// LAB 2: Your code here.
result = nextfree;
nextfree = ROUNDUP((char *) (nextfree+n), PGSIZE);
return result;
```

修改 boot_alloc 函数，boot_alloc(unit32_t n)主要是申请 n 个字节的地址空间，返回申请空间的首地址。由于未初始化的全局变量和静态变量会被自动初始化为 0，系统第一次调用 boot_alloc()这个函数的时候，nextfree 会指向第一个空闲页的首地址。接下来，根据输入的 n，来分配地址。如果 n=0，则返回 nextfree，否则分配 n 字节的地址，返回分配地址的首地址。在整个过程中，需要 4K 对齐。

```
pages = boot_alloc(npages * sizeof (struct PageInfo));
memset(pages, 0, npages*sizeof(struct PageInfo));
```

修改 mem_init()函数，为 pages 申请 npages 的页面，存放这些结构体，并且用 memset 来初始化

```
        size_t i;
        for (i = 0; i < npages; i++) {
                if(i == 0)
                {
                        pages[i].pp_ref = 1;
                        pages[i].pp_link = NULL;
                }
                else if(i>=1 && i<npages_basemem)
                {
                        pages[i].pp_ref = 0;
                        pages[i].pp_link = page_free_list;
                        page_free_list = &pages[i];
                }
                else if(i>=IOPHYSMEM/PGSIZE && i< EXTPHYSMEM/PGSIZE )
                {
                        pages[i].pp_ref = 1;
                        pages[i].pp_link = NULL;
                }

                else if( i >= EXTPHYSMEM / PGSIZE &&   i < ( (int)(boot_alloc(0))
- KERNBASE)/PGSIZE)
                {
                        pages[i].pp_ref = 1;
                        pages[i].pp_link =NULL;
                }
```

```
                        pages[i].pp_ref = 1;
                        pages[i].pp_link =NULL;
                }
                else
                {
                        pages[i].pp_ref = 0;
                        pages[i].pp_link = page_free_list;
                        page_free_list = &pages[i];
                }
        }
```

修改 page_init 函数，系统初始化是分配物理内存

```
if(page_free_list == NULL)
        return NULL;

struct PageInfo* page = page_free_list;
page_free_list = page->pp_link;
page->pp_link = 0;
if(alloc_flags & ALLOC_ZERO)
        memset(page2kva(page), 0, PGSIZE);
return page;
```

```
if(pp->pp_link != 0  || pp->pp_ref != 0)
        panic("page_free is not right");
pp->pp_link = page_free_list;
page_free_list = pp;
return;
```

修改 page_alloc 函数和 page_free 函数，申请和释放页面

Exercise 4. In the file `kern/pmap.c`, you must implement code for the following functions.

```
pgdir_walk()
boot_map_region()
page_lookup()
page_remove()
page_insert()
```

`check_page()`, called from `mem_init()`, tests your page table management routines. You should make sure it reports success before proceeding.

```
int pdeIndex = (unsigned int)va >>22;
if(pgdir[pdeIndex] == 0 && create == 0)
        return NULL;
if(pgdir[pdeIndex] == 0){
        struct PageInfo* page = page_alloc(1);
        if(page == NULL)
        return NULL;
        page->pp_ref++;
        pte_t pgAddress = page2pa(page);
        pgAddress |= PTE_U;
        pgAddress |= PTE_P;
        pgAddress |= PTE_W;
        pgdir[pdeIndex] = pgAddress;
}
pte_t pgAdd = pgdir[pdeIndex];
pgAdd = pgAdd>>12<<12;
int pteIndex =(pte_t)va >>12 & 0x3ff;
pte_t * pte =(pte_t*) pgAdd + pteIndex;
return KADDR( (pte_t) pte );
```

pgdir_walk()：用于返回 va 对应的二级页表的地址(PTE)。给定一个虚拟地址 va 和 pgdir(page director table 的首地址), 返回 va 所对应的 pte(page table entry)。当 va 对应的二级页表存在时，只直接给出 PTE 的地址，当 va 对应的二级页表还没有被创建时，需要手动申请页面并创建，最后返回 PTE 的地址的时，返回 PTE 地址对应的虚拟地址。

```
while(size)
{
   pte_t* pte = pgdir_walk(pgdir, (void* )va, 1);
  if(pte == NULL)
      return;
  *pte= pa |perm|PTE_P;
  size -= PGSIZE;
  pa   += PGSIZE;
  va   += PGSIZE;
}
```

boot_map_region： [va, va+size)映射到[pa, pa+size]

```
pte_t* pte = pgdir_walk(pgdir, va, 0);
if(pte == NULL)
        return NULL;
pte_t pa =  *pte>>12<<12;
if(pte_store != 0)
        *pte_store = pte ;
return pa2page(pa);
```

page_lookup：返回虚拟地址 va 对应的物理地址的页面 page

```
pte_t* pte;
struct PageInfo* page = page_lookup(pgdir, va, &pte);
if(page == 0)
        return;
*pte = 0;
page->pp_ref--;
if(page->pp_ref ==0)
        page_free(page);
tlb_invalidate(pgdir, va);
```

page_remove：将 va 和其对应的页面取消映射

```
pte_t* pte = pgdir_walk(pgdir, va, 1);
if(pte == NULL)
        return -E_NO_MEM;
if( (pte[0] &  ~0xfff) == page2pa(pp))
        pp->pp_ref--;
else if(*pte != 0)
        page_remove(pgdir, va);

*pte = (page2pa(pp) & ~0xfff) | perm | PTE_P;
pp->pp_ref++;
return 0;
```

page_insert():把 va 映射到指定的物理页表 page

> **Exercise 5.** Fill in the missing code in `mem_init()` after the call to `check_page()`.
>
> Your code should now pass the `check_kern_pgdir()` and `check_page_installed_pgdir()` checks.

```
        int perm = PTE_U | PTE_P;
        int i=0;
        n = ROUNDUP(npages*sizeof(struct PageInfo), PGSIZE);
        for(i=0; i<n; i= i+PGSIZE)
                page_insert(kern_pgdir, pa2page(PADDR(pages) + i), (void *)
(UPAGES +i), perm);
```

```
        perm =0;
        perm = PTE_P |PTE_W;
        boot_map_region(kern_pgdir, KSTACKTOP-KSTKSIZE, ROUNDUP(KSTKSIZE,
PGSIZE), PADDR(bootstack), perm);
```

```
int size = ~0;
size = size - KERNBASE +1;
size = ROUNDUP(size, PGSIZE);
perm = 0;
perm = PTE_P | PTE_W;
boot_map_region(kern_pgdir, KERNBASE, size, 0, perm );
```

计算出 pages 结构体的大小，通过 page_insert()进行映射。利用 boot_map_region()实现地址之间的静态映射。

二、实验结果

```
make[1]: Leaving directory '/home/lidongwen/Documents/git/os/lab'
running JOS: (1.0s)
  Physical page allocator: OK
  Page management: OK
  Kernel page directory: OK
  Page management 2: OK
Score: 70/70
```