lab2 pdf

2019年3月29日 2:44

- E1: Pyhysical page management

  pp_ref代表指针的总数，可以用page2pa()把pageinfo*转换为PA

  在一开始作者已经写了npages和npages_basemem，boot_alloc()是为物理地址allocator，page_alloc()是真正的alloc。

  boot_alloc接受参数n，如果n大于0且能分配n个bytes，则分配并返回kva；如果n==0返回下一个空闲页的地址；否则返回NULL

  ```
  if(n > 0){
      char * kva_start = nextfree;
      nextfree = KADDR(PADDR(ROUNDUP(nextfree+n, PGSIZE)));
      return kva_start;
  }else if(n==0){
      return nextfree;
  }
  return NULL;
  ```

  mem_init()只需要写在check_page_free_list(1)之前的部分，这个函数首先申请了PGSIZEbyte大小的内存地址，并对其清0，是page directory初始的page。看注释Allocate an array of npages 'struct PageInfo's and store it in 'pages'. 意思是让我们给pages赋值并清零，查看定义得知pages是一个PageInfo*指针，于是将其当作PageInfo[]数组的起始地址来追踪physical address。

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

  page_init()是要我们初始化pages，也就是上方用来追踪physical address的数组，用它来记录哪些物理地址是空闲的。这里用page_free_list来保留。根据注释内要求，主要分为以下四步：

  1. //Mark physical page[0] as in use.
     for (; i < 1; i++) {
         pages[i].pp_ref = 1;
         pages[i].pp_link = NULL;
     }
  2. //The rest of base memory, [PGSIZE, npages_basemem * PGSIZE) is free
     // page[1]- page[npages_basemem-1]
     for (; i < npages_basemem; i++) {
         pages[i].pp_ref = 0;
         pages[i].pp_link = page_free_list;
         page_free_list = pages + i;
     }
  3. //Then comes the IO hole [IOPHYSMEM, EXTPHYSMEM), which must never be allocated.
     for (i = IOPHYSMEM/PGSIZE; i < (EXTPHYSMEM / PGSIZE); i++) {
         pages[i].pp_ref = 1;

```
        pages[i].pp_link = NULL;
    }
4. //Then extended memory [EXTPHYSMEM, ...).
    for (; i < PGNUM(PADDR(boot_alloc(0))); i++) {
        pages[i].pp_ref = 1;
        pages[i].pp_link = NULL;
    }
5. //rest memory
    for (; i < npages; i++) {
        pages[i].pp_ref = 0;
        pages[i].pp_link = page_free_list;
        page_free_list = pages + i;
    }
```

page_alloc()分配一个新的page，当alloc_flags & ALLOC_ZERO时返回新的页表。

```
    struct PageInfo * ret = NULL;
    if(page_free_list == NULL)
        return NULL;
    if(alloc_flags & ALLOC_ZERO){
        memset(page2kva(page_free_list),0,PGSIZE);
    }
    ret = page_free_list;
    page_free_list = page_free_list->pp_link;
    ret->pp_link = NULL;
    ret->pp_ref = 0;
    return ret;
```

page_free()当且仅当page_decref(pp)时，pp_ref为0才调用，在开始做panic检测，并将该page放入page_free_list以供再次alloc

```
    if(pp->pp_ref != 0)
        panic("page_free:       pp->ref is nonzero!");
    if(pp->pp_link != NULL)
        panic("page_free:       pp->link is not NULL!");
    pp->pp_link = page_free_list;
      page_free_list = pp;
```

至此，完成了part1，手动映射了VA[0, 4MB]-PA[0, 4MB]?

- E3：

使用qemu内置命令，先make qemu-nox，然后ctril-a c进入qemu命令行，info mem即可查看当前内存映射情况。

```
K> QEMU 2.12.0 monitor - type 'help' for more information
(qemu) info mem
0000000000000000-0000000000400000 0000000000400000 -r-
00000000f0000000-00000000f0400000 0000000000400000 -rw
```

question：

Assuming that the following JOS kernel code is correct, what type should variable xhave, uintptr_t or physaddr_t?

```
mystery_t x;
char* value = return_a_pointer();
*value = 10;
```

x = (*mystery_t*) value;
```

uintptr_t是虚拟地址，physaddr_t是物理地址。可以使用KADDR(pa)得到对应的虚地址。直接减去0xf0000000即是物理地址，可以用PADDR(va)来操作

- E4:

**pgdir_walk()**

是说我们要做的是走一个二级页表，返回一个pte_t*

具体实现是：把va前10个bit取出来得到PDE，然后调用page_alloc(ALLOC_ZERO)，并设置ref++。在这之后就能得到一个空的页表（在page_alloc里全部置零了），修改好页目录项再根据va分段算出具体的pte即可。

```
pde_t * target_pde = &pgdir[PDX(va)];
pte_t * target_pt = NULL;
if(!(*target_pde & PTE_P) && create){
        struct PageInfo * pp = (struct PageInfo *)page_alloc(ALLOC_ZERO);
        if(pp == NULL){
                return NULL;
        }
        pp->pp_ref++;
        *target_pde = page2pa(pp)|PTE_P|PTE_W|PTE_U;
}
if(!(*target_pde & PTE_P))
        return NULL;
target_pt = KADDR(PTE_ADDR(*target_pde));
return &target_pt[PTX(va)];
```

boot_map_region()

把va开始的size大小的地址映射到pa开始的size的物理地址上。注意要设置perm | PTE_P位，在这里参数size是PGSIZE的倍数。根据上方TA的hint可知使用pgdir_walk来找到具体的pte

```
int i;
for (i = 0; i < size; i += PGSIZE) {
        pte_t * pte = pgdir_walk(pgdir, (void *) (va + i), 1 );
        *pte = (pa + i) | perm | PTE_P;
}
```

boot_map_region_large()

在这里PDE直接指向页，va段变成了[31..22][21..0]的划分，一个页的大小为4MB，

```
int i;
for (i = 0; i < size; i += 0x400000) {
        pde_t * target_pde = &pgdir[PDX(va+i)];
        if((*target_pde & (PTE_P | PTE_PS)) != (PTE_P | PTE_PS)){
        *target_pde = (pa + i) | perm | PTE_P | PTE_PS;
        }
}
```

page_insert()

同样是把物理地址和虚拟地址映射，成功映射了就pp_ref++，使用pgdir_walk()、page_remove()和page2pa

```
pte_t *pte = pgdir_walk(pgdir, va, 1 );
```
```

```
if (!pte)
        return -E_NO_MEM;
pp->pp_ref++;
if (*pte & PTE_P)
page_remove(pgdir, va);
*pte = page2pa(pp) | perm | PTE_P;
return 0;
```

page_lookup()

返回PageInfo*，使用pa2page()和pgdir_walk()

```
pte_t * pte_address = pgdir_walk(pgdir, va, 0 );
if(pte_store){
*pte_store = pte_address;
}
if(pte_address && (*pte_address & PTE_P))
        return pa2page(PTE_ADDR(*pte_address));
return NULL ;
```

page_remove()

把va对应物理地址解绑，通过page_lookup、 page_decref、 tlb_invalidte来实现

```
pte_t * pte = NULL;
struct PageInfo * page = page_lookup(pgdir, va, &pte);
if(!page)
        return ;
page_decref(page);
tlb_invalidate(pgdir, va);
*pte = 0;
```

- E5:

  总共三个部分需要补充。

  ```
  boot_map_region(kern_pgdir,UPAGES, PTSIZE, PADDR(pages), PTE_U);
  boot_map_region(kern_pgdir,KSTACKTOP-KSTKSIZE, KSTKSIZE , PADDR(bootstack), PTE_W);
  boot_map_region_large(kern_pgdir,KERNBASE, -KERNBASE, 0, PTE_W);
  ```

  question:

  1. What entries (rows) in the page directory have been filled in at this point? What addresses do they map and where do they point? In other words, fill out this table as much as possible:

Entry	Base Virtual Address	Points to logically
1023	0xffc00000	page table for top 4MB of phys memory
1022	0xff800000	
960	0xf0000000	KERNBASE
.		.
956	0xef000000	UPAGES
2	0x00800000	user programs data & heap
1	0x00400000	UTEMP
0	0x00000000	手动映射的4M物理地址

2. 靠页表里的PTE_Ubit位，在mmu处理虚拟地址到内核地址的时候会检验它
3. npages = 0x40ff, PGSIZE=0x1000,page有40MB。总共地址有 0x400000/sizeof(PageInfo)*PGSIZE = 2GB，所以最大为2G
4. 有三部分，用于转换的PageDirectory,PageTable和用于记录和引用的pages。可以使用largepage来减少开销。
5. 因为page模式下cpu到mmu的全部是虚拟地址，没有手动映射0-4MB的话会找不到对应物理地址而出错。

- Challenge：
  完成了showmappings指令的添加，通过pgdir和va先获取PDE，再根据PDE遍历输出PTE。
  测试样例：
  showmappings 0xef000000 0xf0000000
  竖直方向上按照PDE [PTE.....PTE] ;PDE [PTE...PTE]的格式输出
  每一行按照[virtual addr physical addr entry permission]的格式输出