PA3-3 实验报告

191220163 计算机科学与技术系 张木子苗

1. Kernel的虚拟页和物理页的映射关系是什么？请画图说明；

   kernel/src/memory/kvm.c 中有这样一段代码：

```
PDE *pdir = (PDE *)va_to_pa(kpdir);
PTE *ptable = (PTE *)va_to_pa(kptable);
uint32_t pdir_idx, ptable_idx, pframe_idx;

/* make all PDE invalid */
memset(pdir, 0, NR_PDE * sizeof(PDE));

/* fill PDEs and PTEs */
pframe_idx = 0;
for (pdir_idx = 0; pdir_idx < PHY_MEM / PT_SIZE (2^5); pdir_idx++)
{
    pdir[pdir_idx].val = make_pde(ptable);
    pdir[pdir_idx + KOFFSET / PT_SIZE].val = make_pde(ptable);
    for (ptable_idx = 0; ptable_idx < NR_PTE; ptable_idx++)
    {
        ptable->val = make_pte(pframe_idx << 12);
        pframe_idx++;
        ptable++;
    }
}
```

   根据前面所定义的数值，可知 PHY_MEM / PT_SIZE = 2^{27} / 2^{22} = 32， $KOFFSET$ / PT_SIZE = 0xC0000000 >> 22 = 0x300，NR_PTE = 2^{10}

   make_pde和make_pte 是这样的宏：

```
#define make_pde(addr) ((((uint32_t)(addr)) & 0xfffff000) | 0x7)
#define make_pte(addr) ((((uint32_t)(addr)) & 0xfffff000) | 0x7)
```

   它确保这个宏的值的低 3 位都为 1，中间 9 位都为 0，而高 20 位保留其参数高20位原来的值。

   所以只有当 ptable 的高 20 位改变时，make_pde(ptable) 的值才会改变。

   对于内层的 for 循环，每次循环 ptable++。每一次外层循环中，内层循环一共进行 2^{10} 次，ptable一共自增 2^{10} 次，而因为ptable为指针，所以实际上的数值自增了2^{12} 。所以外层循环每进行1次，ptable 的高20位会加1。

   而每次外层循环中，又有

```
pdir[pdir_idx].val = make_pde(ptable);
pdir[pdir_idx + KOFFSET / PT_SIZE].val = make_pde(ptable);
```

   所以页目录表的的第pdir_idx项和第pdir_idx + 0x300项都被映射到同一个页表

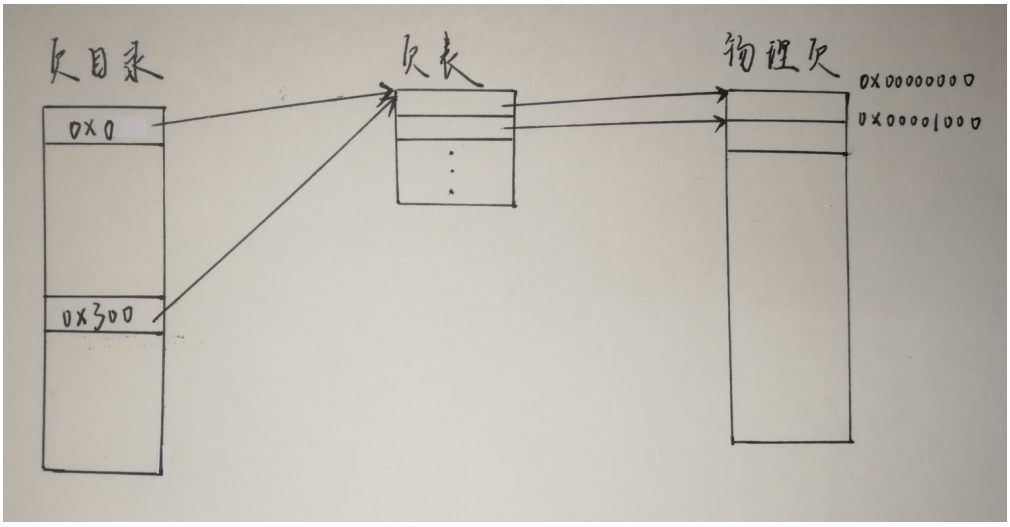
   至于虚拟页到物理页的映射，我们用该语句实现

```
ptable->val = make_pte(pframe_idx << 12);
pframe_idx++;
ptable++;
```

pframe_idx被初始化为0，左移12位之后得到0x00000000，为第0个物理页的首地址，

之后pframe_idx++，左移12位之后得到0x00001000，为第1个物理页的首地址，

画出图来如下所示：



所以，已知Kernel的代码从虚拟地址0xC0030000 开始，其页目录项为0x300，它被映射到与页目录项为0x0相同的页表。根据上图中的映射关系，可知：Kernel的虚拟页和物理页的映射关系是：

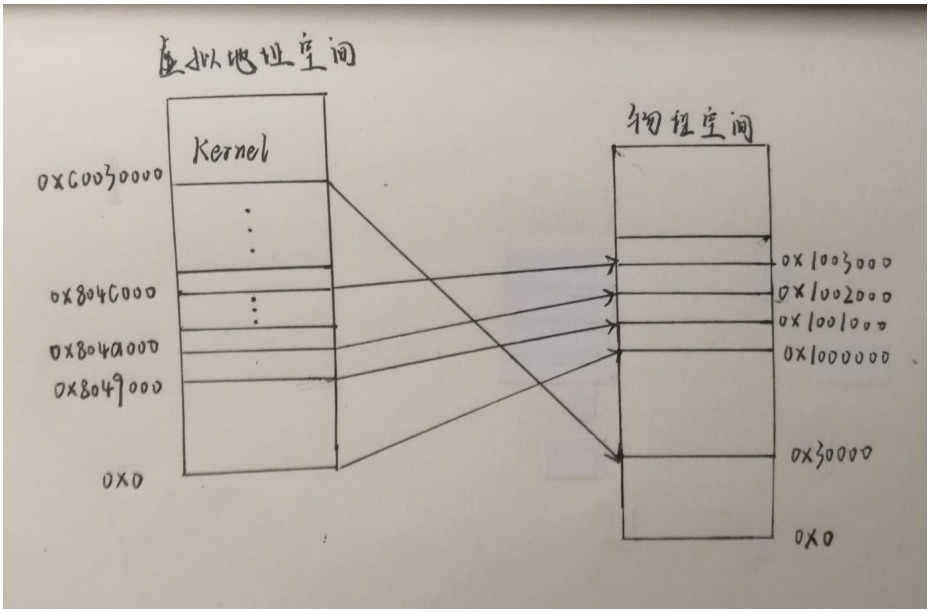
0xC0030000 -> 0x00030000；0xC0031000 -> 0x00031000；之后的以此类推。

2. 以某一个测试用例为例，画图说明用户进程的虚拟页和物理页间映射关系又是怎样的？Kernel映射为哪一段？你可以在`loader()`中通过`Log()`输出`mm_malloc`的结果来查看映射关系，并结合`init_mm()`中的代码绘出内核映射关系。

   在`loader()`中通过`Log()`输出`mm_malloc`的结果如下：

```
./nemu/nemu --kernel --testcase mov-c
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/mov-c
(nemu) c
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr = 0x0, paddr = 0x1000000

nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr = 0x8049000, paddr = 0x1001000

nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr = 0x804a000, paddr = 0x1002000

nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr = 0x804c000, paddr = 0x1003000

nemu: HIT GOOD TRAP at eip = 0x08049103
NEMU2 terminated
```

   由上图我们可以很容易得到以下映射关系：



在上一问中已经知道，kernel被映射到物理地址为0x30000的页面

3. “在Kernel完成页表初始化前，程序无法访问全局变量”这一表述是否正确？在`init_page()`里面我们对全局变量进行了怎样的处理？

   这个说法是不正确的。

   没有页表，我们不能够进行线性地址对物理地址的转换，但是可以直接用物理地址访问全局变量。

   比如说在start.S中，我们利用的便是宏定义va_to_pa(x)

   ```
   #ifdef IA32_PAGE
   #   define KOFFSET 0xc0000000
   #   define va_to_pa(x) (x - KOFFSET)
   #else
   #   define va_to_pa(x) (x)
   #endif
   ```

   从align to page可知，在`init_page()`里面我们对全局变量进行了按页对齐的处理。

   ```
   PDE kpdir[NR_PDE] align_to_page;                // kernel page directory
   PTE kptable[PHY_MEM / PAGE_SIZE] align_to_page; // kernel page tables
   ```