PA_3-3 实验报告

计算机科学与技术系 张桓 191220156

1.Kernel 的虚拟页和物理页的映射关系是什么？请画图说明。

答：Kernel 的页表在 kernel/src/main.c 源文件中的 init()函数里，通过调用 init_page()函数来初始化 Kernel 的页表。找到 init_page()函数在 kernel/src/memory/kvm.c 中，代码如下：

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

根据数值的宏可以知道：PHY_MEM / PT_SIZE = 2^27 / 2^22 = 32

KOFFSET / PT_SIZE = 0xc0000000 / 2^22 = 0x300, NR_PTE = 2^10

make_pde 的宏定义为：

#define make_pde (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，高 20 位保留其参数原来的值。所以只有当 ptable 的高 20 位改变时，

make_pde(ptable) 的值才会改变。

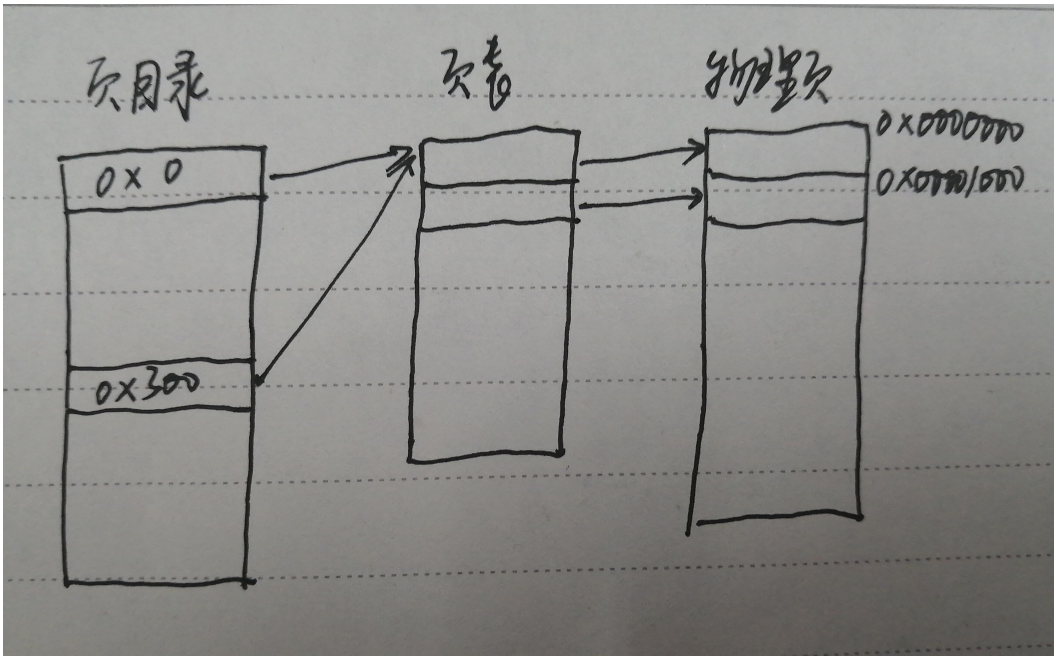
可以看到代码通过 for 循环实现对 Kernel 的页目录和页表的初始化，每一次外层循环内层循环重复 2^10 次，每次都有 ptable++，一共增加 2^10。而 ptable 是按照 2^12 对齐的，每次外循环中都有

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

可以看到页目录表的第 pdir_idx 项和第 pdir_idx + 0x300 都被映射到同一个页表。从虚拟页到物理页的映射，由以下代码实现：

```
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，映射图如下：



所以，已知 Kernel 的代码从虚拟地址 0xC0000000 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：

0xC0000000 -> 0x00000000；0xC0001000 -> 0x00001000；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的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。

答：在 loader()函数中加入 Log()函数输出 mm_malloc()的结果，代码如下：

```
paddr = mm_malloc(ph->p_vaddr, ph->p_memsz);
Log("vaddr:%x paddr:%x", ph->p_vaddr, paddr);
```

输出加载页起始的虚拟地址和对应的物理地址，我们选 add 测试用例观察输出如下图：

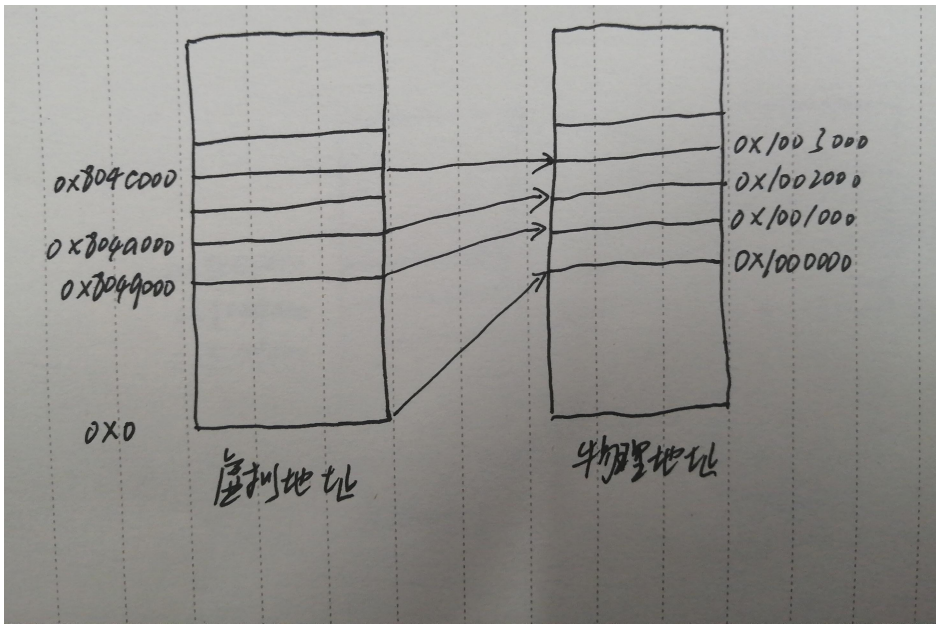
```
./nemu/nemu --autorun --testcase add --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/add
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,30,loader] {kernel} ELF loading from ram disk.
nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr:0 paddr:1000000
nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr:8049000 paddr:1001000
nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr:804a000 paddr:1002000
nemu trap output: [src/elf/elf.c,45,loader] {kernel} vaddr:804c000 paddr:1003000
nemu: HIT GOOD TRAP at eip = 0x08049094
```

可以看到测试用例中以 0X0 为起始地址的虚拟页映射到起始地址为 0X1000000 的物理页；以 0X8049000 为起始地址的虚拟页映射到起始地址为 0X1001000 的物理页；以 0X804a000 为起始地址的虚拟页映射到起始地址为 0X1002000 的物理页；以 0X804c000 为起始地址的虚拟页映射到起始地址为 0X1003000 的物理页.

观察 init_mm()函数代码如下：

```
PDE *kpdir = get_kpdir();
/* make all PDE invalid */
memset(updir, 0, NR_PDE * sizeof(PDE));
/* create the same mapping above 0xc0000000 as the kernel mapping does */
memcpy(&updir[KOFFSET / PT_SIZE], &kpdir[KOFFSET / PT_SIZE],
         (PHY_MEM / PT_SIZE) * sizeof(PDE));
```

kernel 的映射从 0x100300 开始，映射关系如下图：



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做了哪些处理？

答：不正确，如果当前进程未进入保护模式而是实模式，不需要页表就可以直接通过逻辑地址(即物理地址)访问到数据，这时程序也可以访问全局变量，而页表此时并没有被初始化。

在 init_page()中完成了页目录和页表的初始化，对全局变量进行了按页对齐处理。