PA3 实验报告

161220039 冯诗伟

PA3-2 保护模式

1. NEMU在什么时候进入了保护模式?

答:

```
2         lgdt    va_to_pa(gdtdesc) # See i386 manual for more information
3         movl    %cr0, %eax        # %CR0 |= PROTECT_ENABLE_BIT
4         orl     $0x1, %eax
5         movl    %eax, %cr0
```

在kernel/start/start.S中这几句汇编指令，将cr0寄存器的最低位pe设为1，从而开启保护模式。

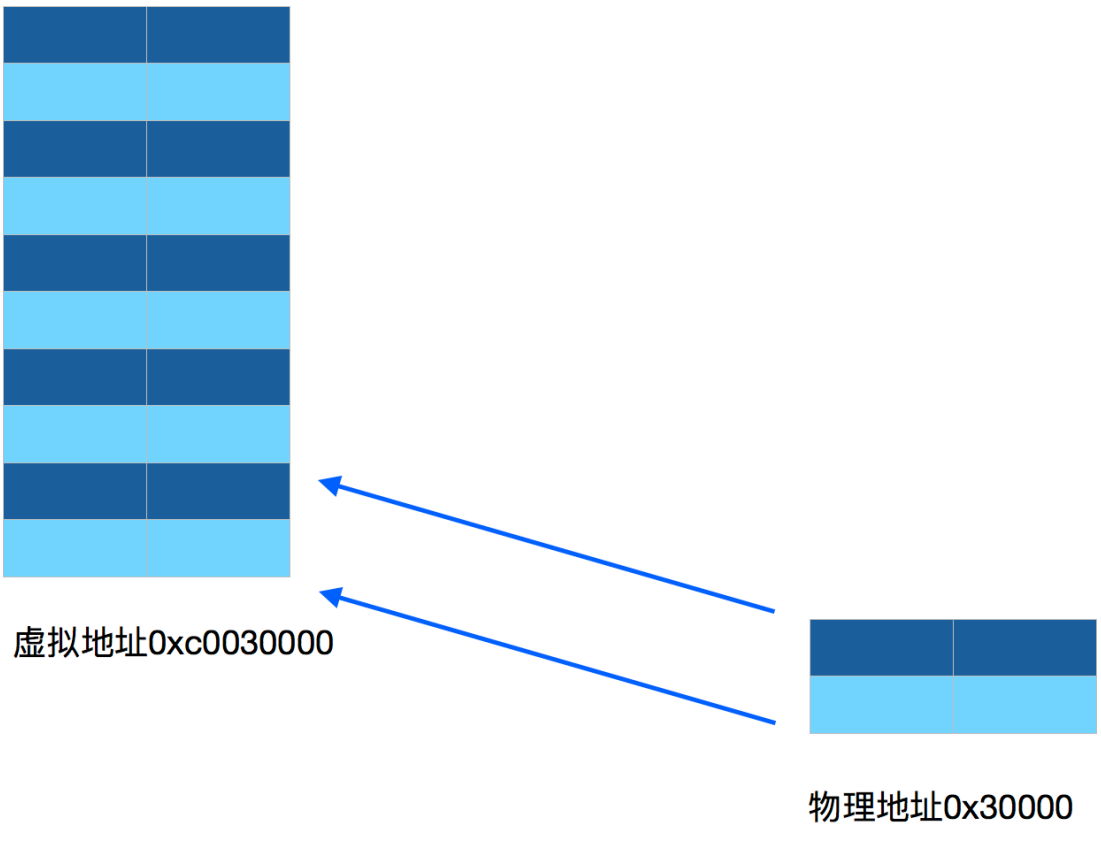
2. 在GDTR中保存的段表首地址是虚拟地址、线性地址、还是物理地址？为什么?

答：GDTR中保存的段表首地址是线性地址。gdtr.base是32位，所以不是虚拟地址；而且现在还没有开始分页，所以也不是物理地址。

PA3-3 虚拟地址转换

1. Kernel的虚拟页和物理页的映射关系是什么？请画图说明；

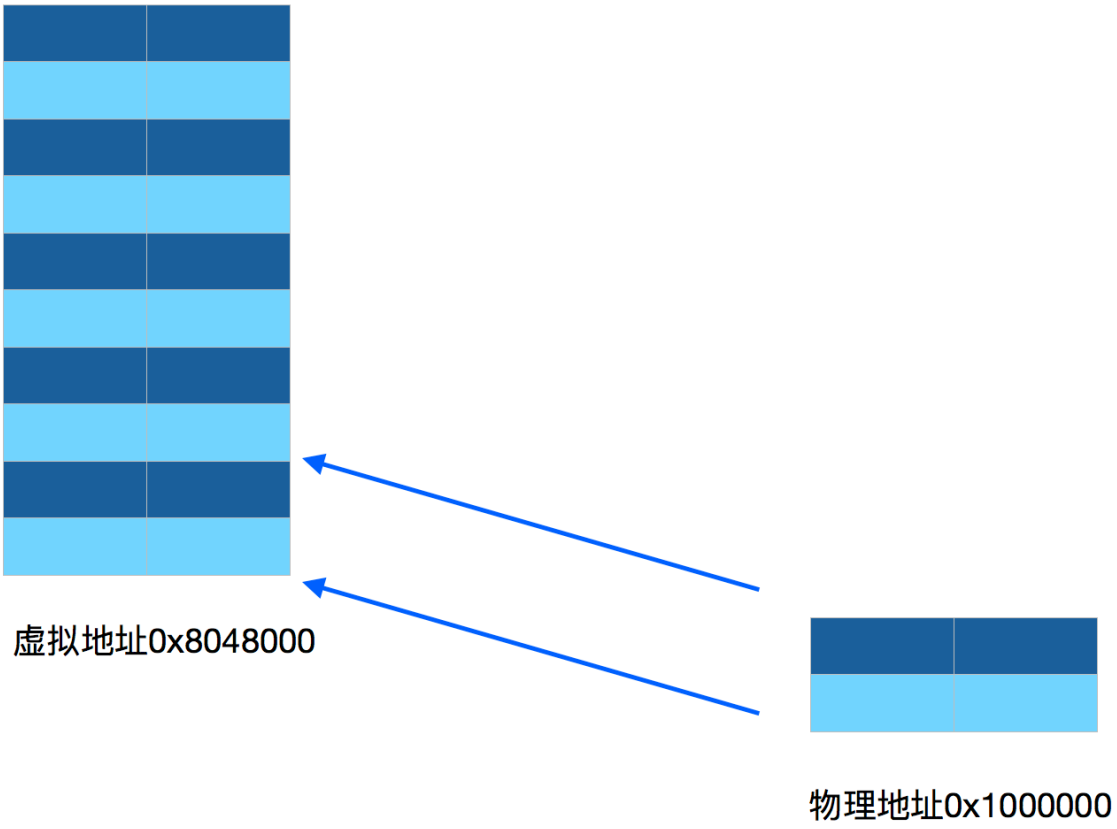
答：物理页的起始地址加上0xC0000000，就会得到虚拟页的起始地址。



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。
答：

```
Execute ./kernel/kernel.img ./testcase/bin/mov-c
nemu trap output: [src/main.c,76,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,32,loader] {kernel} ELF loading from ram disk.
nemu trap output: vaddr=0x8048000
nemu trap output: paddr=0x1000000
nemu trap output: vaddr=0x804a000
nemu trap output: paddr=0x1001000
nemu: HIT GOOD TRAP at eip = 0x08048197
NEMU2 terminated
```

以mov-c为例，用户进程的物理页从0x01000000开始，虚拟页从0x08048000开始。



在init_mm()函数中，kernel映射到Koffset=0xC0000000以上的内核区

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?
答：正确。

```
/* set up page tables for kernel */
void init_page(void) {
    CR0 cr0;
    CR3 cr3;
    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 ++;

        }
    }
```

kernel的页目录表和页表

页目录表记录页表首地址

填充页表内容

在init_page()里，将每一个页目录项与对应的页表绑定，同时将对应的虚拟页绑定；接着把每一项页表与对应的物理页绑定起来。在调用init_page()之前，全局变量都在内核区，用户程序无法访问。

实验过程中遇到的一些问题

PA3-1

认知上的错误：

在从内存向cache中写内容时，应考虑把内存地址的低6位（即表示块内偏移量的部分）去掉，即paddr&0x7ffffc0。因为从主存拷贝一整块到cache中，需要考虑这一整块的开始地址，而不是要访问的块中的某处偏移地址。要访问的内容应等待cache整行被更新完之后，再根据块内偏移量来访问。

编程上：起变量名一定要起简明扼要的名字，否则可读性太差，难以调试。

注意代码的复用性（千万不要复制粘贴⚠️）

PA3-2

1. 注意位存储的顺序，位存储是从右向左存储在结构体中
2. 读写时字节起始位置的顺序

PA3-3

1. 在写关于控制寄存器的mov指令时，从modrm字节中解析出控制寄存器的地址刚开始有些疏忽。

2. 之前在PA2-1中写指令时，用到了paddr_read/write，分页之后会出问题，所以把PA2-1中的paddr改成了laddr/vaddr。