PA3 实验报告

161220096 欧阳鸿荣

3.2

1. NEMU 在什么时候进入了保护模式？

```
        .globl start
        start:
#ifdef IA32_INTR
                cli
#endif

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

        # Complete transition to 32-bit protected mode by using long jmp
        # to reload %CS and %EIP.  The segment descriptors are set up with no
        # translation, so that the mapping is still the identity mapping.
                ljmp    $GDT_ENTRY(1), $va_to_pa(start_cond)

        start_cond:
        # Set up the protected-mode data segment registers
                movw    $GDT_ENTRY(2), %ax
                movw    %ax, %ds          # %DS = %AX
                movw    %ax, %es          # %ES = %AX
                movw    %ax, %ss          # %SS = %AX

        # Set up a stack for C code.
                movl $0, %ebp
                movl $(128 << 20), %esp
                sub $16, %esp
                jmp init                                  # never return

        # GDT
        .p2align 2                      # force 4 byte alignment
        gdt:
                MAKE_NULL_SEG_DESC                        # empty segment
```

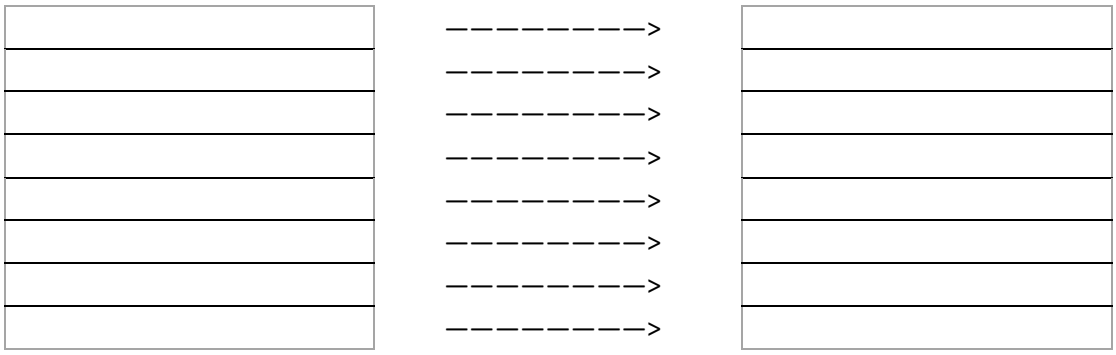
如图所示，在执行玩红框中的这段汇编代码之后，控制寄存器 cr0 被赋值，从而控制是否进入保护模式的 PE 位也被置为 1，从这时候起，NEMU 就进入了保护模式。

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？

在 GDTR 中保存的段表首地址是线性地址。因为在开启分段后，是通过 GDTR 中保存的段表首地址加上偏移量来找到对应得段表的线性地址的，然后才经过一系列转换得到虚拟地址。如果段表首地址是虚拟地址，那么就会陷入一种“通过虚拟地址找虚拟地址”的矛盾中，这显然得不到正确的地址。

3.3

1. Kernel 的虚拟页和物理页的映射关系是什么？请画图说明；



从 0xc0030000 开始一页 4KB　　从 0x00030000 开始一页 4KB

Kernel 虚拟页　　**物理页**

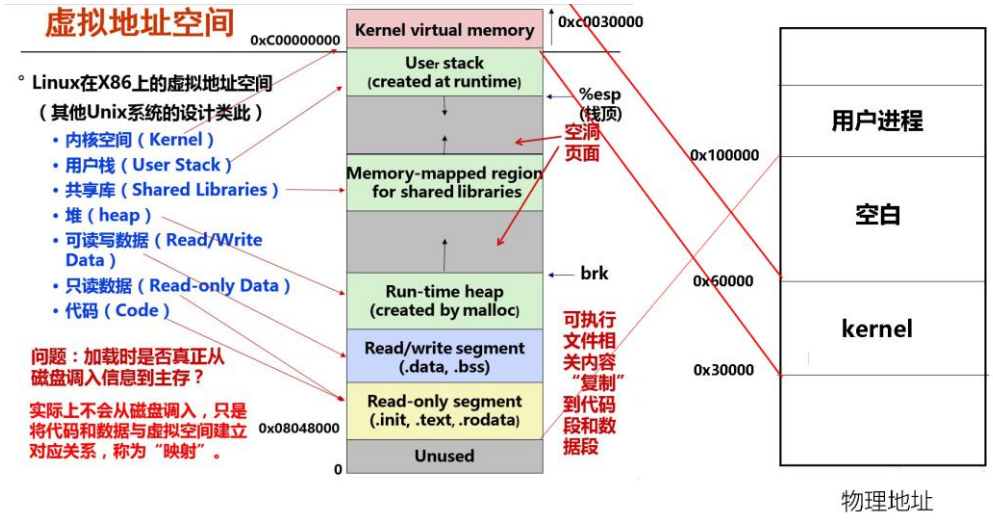
Kernel 的虚拟页和物理页一一对应

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。

```
Execute ./kernel/kernel.img ./testcase/bin/mov-c
Welcome to cache!

nemu trap output: [src/main.c,75,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,43,loader] {kernel} before = 0x8048000
nemu trap output: [src/elf/elf.c,46,loader] {kernel} after = 0x1000000
nemu trap output: [src/elf/elf.c,43,loader] {kernel} before = 0x804a000
nemu trap output: [src/elf/elf.c,46,loader] {kernel} after = 0x1001000
nemu: HIT GOOD TRAP at eip = 0x08048197
NEMU2 terminated
```

在 loader()中通过 Log()输出 mm_malloc 的结果来查看映射关系，可见是把从 0x8048000 开始的一段数据映射到从 0x1000000 开始的物理地址中，映射关系图如下



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？

这个说法正确。实地址模式下，全局变量在存储在高于 0x30000 的位置。因为在页表初始化前，全局变量存储在高于 0xc000000 的位置，因此在初始化页表前程序无法访问到这个地址

```
/* 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 ++;

        }
}
```

init_page()中，实际创建了两个页表，一个加上 KOFFSET（也即 0xc0000000），一个不加上 0xc0000000，这样使得我们在初始化页表之后，可以访问到存储位置高于 0xc0000000 的全局变量。