
南京大学

# PA-3-3：分页机制&虚拟地址转换

课程名称：计算机系统基础

姓名：孙文博

学号：201830210

邮箱：201830210@smail.nju.edu.cn

实验时间：2022.6.1 - 2022.6.14

# 一、 实验进度

1. 了解分页机制，页表，线性地址与物理地址的转换等背景知识；
2. 修改 include/config.h 头文件，为 NEMU 开启分页机制；
3. 修改 kernel/Makefile，将 LDFLAGS 中的-Ttext=0x30000 修改为虚拟地址空间中的位置-Ttext=0xC0030000；
4. 修改 testcase/Makefile，去掉 LDFLAGS 中之前对于-Ttext 的设置，使得测试用例的的只读数据和代码段从虚拟地址 0x8048000 以上开始，符合 Linux 虚拟地址空间的约定。
5. 在 nemu/include/cpu/reg.h 头文件中，为 CPU_STATE 结构添加上 CR3 寄存器：

```
45  typedef union
46  {
47      struct
48      {
49          uint16_t reserve:12;
50          uint32_t pdir:20;
51      };
52      uint32_t val;
53  }CR3;
54
```

6. 修改 laddr_read() 和 laddr_write() 进行地址转换：

```
43  uint32_t laddr_read(laddr_t laddr, size_t len)
44  {
45      assert(len == 1 || len == 2 || len == 4);
46
47      if(cpu.cr0.pg == 1)
48      {
49          if((laddr >> 12) != ((laddr + len - 1) >> 12))
50          {
51              uint32_t ret = 0;
52              uint32_t num = laddr&0xfff;
53              uint32_t temp = 4096-num;
54              ret = paddr_read(page_translate(laddr), temp);
55              ret+=((paddr_read(page_translate(laddr + temp), len - temp))<<(8*temp));
56              return ret;
57          }
58          else
59          {
60              paddr_t paddr = page_translate(laddr);
61              return paddr_read(paddr, len);
62          }
63      }
64      else
65          return paddr_read(laddr, len);
66  }
```

```
68 void laddr_write(laddr_t laddr, size_t len, uint32_t data)
69 {
70     assert(len==1||len==2||len==4);
71     if(cpu.cr0.pg==1)
72     {
73         if((laddr >> 12) != ((laddr + len - 1) >> 12))
74         {
75             uint32_t num = laddr & 0xfff;
76             uint32_t temp = 4096 - num;
77             paddr_write(page_translate(laddr), temp, data);
78             paddr_write(page_translate(laddr + temp), len - temp, data >> (8 * temp));
79         }
80         else
81         {
82             paddr_t paddr = page_translate(laddr);
83             paddr_write(paddr, len, data);
84         }
85     }
86     else
87         paddr_write(laddr, len, data);
88 }
```

7. 补全 memory/mmu/page.c 文件中 page_translate()函数：

```
1  #include "cpu/cpu.h"
2  #include "memory/memory.h"
3
4  // translate from linear address to physical address
5  paddr_t page_translate(laddr_t laddr)
6  {
7  #ifndef TLB_ENABLED
8      //printf("\nPlease implement page_translate()\n");
9      //fflush(stdout);
10     //assert(0);
11     uint32_t dir=(laddr>>22)&0x3ff;
12     uint32_t page=(laddr&0x3ff000)>>12;
13     uint32_t offset=laddr&0xfff;
14     PDE pde;
15     uint32_t addr_pde=(cpu.cr3.pdir<<12)+4*dir;
16     memcpy(&pde.val,(void*)(hw_mem+addr_pde),4);
17     if(pde.present==0)
18     {
19         printf("\n%x\n",cpu.eip);
20     }
21     assert(pde.present==1);
22     uint32_t addr_pte=(pde.page_frame<<12)+page*4;
23     PTE pte;
24     memcpy(&pte.val,(void*)(hw_mem+addr_pte),4);
25     assert(pte.present==1);
26     return (pte.page_frame<<12)+offset;
27 #else
28     return tlb_read(laddr) | (laddr & PAGE_MASK);
29 #endif
30 }
31
```

8. 修改 Kernel 的 loader()，使用 mm_malloc 来完成对用户进程空间的分配：

```
42  #ifndef IA32_PAGE
43              paddr=ph->p_vaddr;
44  #else
45              paddr=mm_malloc(ph->p_vaddr,ph->p_memsz);
46  #endif
```

9. 最后，执行 make test_pa-3-3 命令并通过各测试用例：

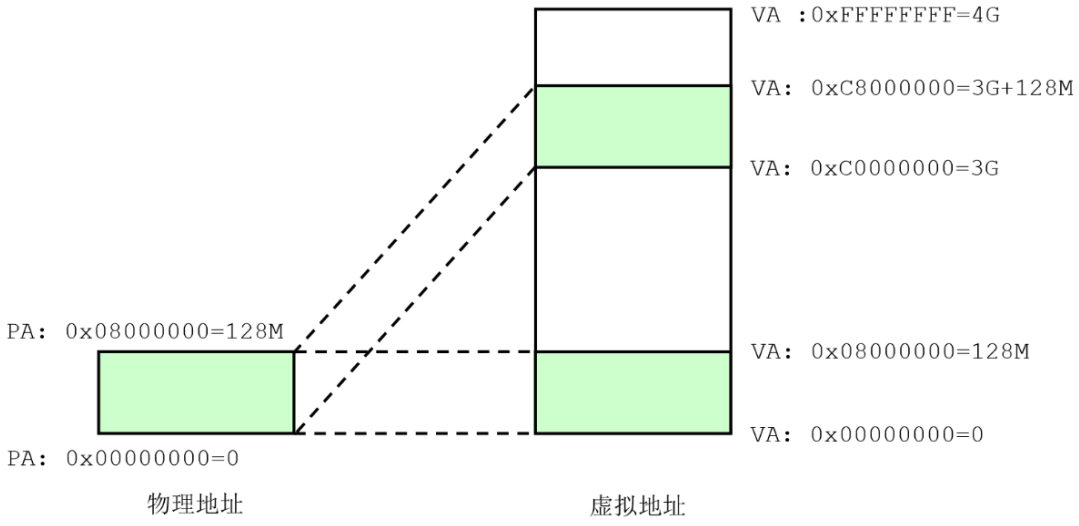
```
./nemu/nemu --autorun --testcase struct --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/struct
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu: HIT GOOD TRAP at eip = 0x0804910c
NEMU2 terminated
./nemu/nemu --autorun --testcase string --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/string
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: HIT GOOD TRAP at eip = 0x0804916a
NEMU2 terminated
./nemu/nemu --autorun --testcase hello-str  --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-str
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: HIT GOOD TRAP at eip = 0x08049105
NEMU2 terminated
./nemu/nemu --autorun --testcase test-float --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/test-float
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: HIT BAD TRAP at eip = 0x080490c8
NEMU2 terminated
make-[1]: Leaving directory '/home/pa201830210/pa_nju'
pa201830210@edb32e250119:~/pa_nju$
```

## 二、 思考题

1. Kernel 的虚拟页和物理页的映射关系是什么？请画图说明；

答：每个进程都有自己独立的页表来描述该进程的虚拟地址空间和物理地址空间之间的映射关系。从较为直观的角度来说，一个 32 位的线性地址可以分为两个部分：高 20 位指出该线性地址属于哪一个虚拟页，而低 12 位则指出该地址具体指向的字节在该

虚拟页内的偏移量是多少。若参照段表的组织方式，将所有 2^20 个页表项都按顺序存放在一个数组中，那么地址转换的过程就可以简单地表达为：使用线性地址中的高 20 位作为索引，查找页表；提取对应页表项中的物理页号，加上线性地址中低 12 指出的页内偏移量，便能够获取对应的物理地址。其映射关系如图所示：




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：以 mov-c 测试文件为例，通过 log() 函数输出的信息分析：

```
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/mov-c
nemu trap output: [src/memory/mm.c,31,init_mm] {kernel} updir: 48049000
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,43,loader] {kernel} paddr: 1000000, vaddr: 0
nemu trap output: [src/elf/elf.c,43,loader] {kernel} paddr: 1001000, vaddr: 8049000
nemu trap output: [src/elf/elf.c,43,loader] {kernel} paddr: 1002000, vaddr: 804a000
nemu trap output: [src/elf/elf.c,43,loader] {kernel} paddr: 1003000, vaddr: 804c000
```

根据上图所示，虚拟地址 0x0 映射到物理地址 0x1000000，虚拟地址 0x8049000 映射到物理地址 0x1001000，虚拟地址 0x804a000 映射到物理地址 0x1002000，虚拟地址 0x804c000 映射到物理地址 0x1003000。kernel 从虚拟地址 0xc0030000 开始，映射到物理地址 0x30000 开始。

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。因为在实地址模式下，全局变量存储在高于 0x30000 的位置，而在页表初始化前，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。在 init_page()中，实际创建了两个页表，一个加上 KOFFSET（即 0xc0000000），一个不加 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。如下图所示：

```
38 #ifdef IA32_PAGE
39     asm volatile("jmp *%0"
40                  :
41                  : "r"(init_cond + 0xc0000000));
42 #else
43     asm volatile("jmp *%0"
44                  :
45                  : "r"(init_cond));
46 #endif
47
48     /* Should never reach here. */
49     nemu_assert(0);
```

## 三、 总结与反思

本阶段的内容与书本知识联系比较紧密，涉及分页机制，快表和页表，线性地址到主存地址的转换等。同时临近考试周，也相当于是对 ICS 课本知识的复习和巩固了。希望这周能够顺利通过考试，全身心投入 PA-4 阶段的研究!