

exam

在自映射的条件下，请实现函数完成下列任务：

任务0

64位操作系统采用三级页表进行虚拟内存管理，每个页表大小为4KB，页表项需要字对齐，其余条件与二级页表管理32位操作系统相同。请问64位中最少用多少位表示虚拟地址。

任务1

输入二级页表的起始虚拟地址va，返回一级页表的起始虚拟地址。

任务2

输入页目录的虚拟地址va和一个整数n，返回页目录第n项所对应的二级页表的起始虚拟地址。

上面的任务1与2，是让你熟悉自映射的有关知识，所有的地址都只是一个u_long类型的数字，并没有 和操作系统打交道，那么最后一个任务则要求你真正填写页表。

任务3

给定一个一级页表的指针pgdir和二级页表起始虚拟地址va，va为内核态虚拟地址。把合适的地址填写到pgdir的指定位置，使得pgdir能够完成正确的自映射。（即计算出va对应的物理地址所在一级页表项位置，并在那里填入正确的页号和权限位）

输入输出约定：

在include/pmap.h中声明，同时在mm/pmap.c中编写函数：

```
u_long cal_page(int func, u_long va, int n, Pde *pgdir);
```

输入：

func为0，1，2，3分别对应前面的任务0123。

va为前述任务中的虚拟地址，func为0时，传入0。

n仅在第二项任务中有意义，意义同题目叙述。在func为0，1，3时，传入0。

pgdir仅在第三项任务中有意义，意义同题目叙述。在func为0，1，2时，传入0。

输出：

任务0要求返回正确答案，任务1，2 返回要求地址，任务3返回0即可。

解答

这次测评非常坑，必须要把任务0做出来，否则测评机会反馈整个exam零分。而这一点是离结束40分内才说的，有的教室甚至没说，所以有的放弃任务0而去de其他任务的bug的同学（别骂了），血亏。

任务0答案是39，一个页表4KB，64位机中，一个entry大小为64b=8B，一个页表中有 2^{19} 项，三级页表， $total_bit = 3 \times 9 + pgshift = 3 \times 9 + 12 = 39$ 。

Extra

请实现满足下列要求的函数：

给定一个页目录的起始地址，统计在相应的页表中使用的物理页面的情况，其中需要对传入的cnt数组进行修改，使cnt[i]表示第i号物理页被页目录下的虚拟页映射的总次数。

要求

在pmap.c文件中编写函数 int count_page(Pde *pgdir, int *cnt)

在pmap.h文件中进行函数声明 int count_page(Pde *pgdir, int *cnt);

函数输入的**Pde**指针的值为页目录的内核虚拟地址，**cnt**为数组首地址，函数的返回值为**cnt**数组的元素个数，即物理页的数量（在我们的操作系统中，这个的值为一个常量），**cnt[i]**表示页目录下有**cnt[i]**个虚拟页映射到了第**i**号物理页。

注意

如果想本地测试的话可以在**init.c**中进行测试，提交时会进行替换。

自己写的其他测试辅助函数不要有**standard**单词，防止和评测冲突导致编译错误。

提示

物理页的使用情况包括页目录、二级页表及所有被映射到的物理页。

一个物理页可能被进程的多个虚拟页映射。

传入的**cnt**数组不一定全0。

```
u_long cal_page(int func, u_long va, int n, Pde *pgdir) {
    u_long second_begin, first_pn;
    switch (func) {
        case 0:
            return 39;
            break;
        case 1:
            return va + (va >> 10);
            break;
        case 2:
            second_begin = va & (0 - (1 << 22));
            return second_begin + (n << 12);
            break;
            KADDR(PTE_ADDR(*(va+4n)))?
        case 3:
            first_pn = (va >> 22);
            *(pgdir + first_pn) = PADDR(pgdir) | PTE_V | PTE_R;
            return 0;
            break;
            *pgdir=PADDR(pa) | PTE_V | PTE_R;?
    }
    return 0;
}
```

```
void count_page(Pde *pgdir, int *cnt,int size)
{
    Pde *pgdir_entry;
    Pte *pgtab, *pgtab_entry;
    int i, j;
    for (i = 0; i < size; i++)
    {
        cnt[i] = 0;
    }
    cnt[PPN(PADDR(pgdir))]+=;
    for (i = 0; i < PTE2PT; i++)
    {
        pgdir_entry = pgdir + i;
        if((*pgdir_entry)&PTE_V)
        {
            cnt[PPN(*pgdir_entry)]++;
            pgtab = KADDR(PTE_ADDR(*pgdir_entry));
            for (j = 0; j < PTE2PT; j++)
            {
                pgtab_entry = pgtab + j;
```

```

        if ((*pgtab_entry) & PTE_V)
        {
            cnt[PPN(*pgtab_entry)]++;
        }
    }
}
return;
}

```

构建方法

1. 给定一个页表基址 PT_{base} ，该基址需4M对齐，即：

$$PT_{base} = ((PT_{base}) \gg 22) \ll 22;$$

不难看出， PT_{base} 的低22位全为0。

2. 页目录表基址 PD_{base} 在哪里？

$$PD_{base} = PT_{base} | (PT_{base}) \gg 10$$

3. 自映射目录表项 $PDE_{self-mapping}$ 在哪里？

$$PDE_{self-mapping} = PT_{base} | (PT_{base}) \gg 10 | (PT_{base}) \gg 20$$

北京航空航天大学

计算机学院

OS教学组

18

page2kva(struct Page **pp*) 得到页pp的虚地址

page2ppn(struct Page **pp*) 得到页pp的物理页号

page2pa(struct Page **pp*) 得到页pp的实地址

pa2page(u_long *pa*) 得到实地址pa的页

PPN(*va*) 得到实地址va的物理页号

PDX(va) 可以获取虚拟地址 va 的 31-22 位，PTX(va) 可以获取虚拟地址 va 的 21-12 位。

CPU 发出的地址均为虚拟地址，因此获取相关物理地址后，需要转换为虚拟地址再访问。对页表进行操作时处于内核态，因此使用宏 KADDR 即可。

PTE_ADDR(pde) 后12位清零，得到二级页表物理地址

20 #define PTE_G 0x0100 // 全局位

21 #define PTE_V 0x0200 // 有效位

22 #define PTE_R 0x0400 // 修改位，如果是0表示只对该页面进行过读操作，否则进行过写操作，要引发中断将内容写回内存

23 #define PTE_D 0x0002 // 文件缓存的修改位dirt

在进行内存初始化时，mips_detect_memory()、mips_vm_init()与page_init()被依次调用。

mips_detect_memory()用来初始化一些全局变量（此处将物理内存大小设置为64MB，在实际中，内存大小是由硬件得到的，这里只是模拟了检测物理内存大小这个过程）。其余的函数的功能为：

static void *alloc(u_int n, u_int align, int clear): 申请一块内存，返回首地址。

static Pte *boot_pgdir_walk(Pde *pgdir, u_long va, int create): 从页目录项中找出虚拟地址va对应的页表项，若create置位，则不存在时创建。

void boot_map_segment(Pde *pgdir, u_long va, u_long size, u_long pa, int perm):

将虚拟地址 (va, va+size-1) 映射到物理地址 (pa, pa+size-1)。

void mips_vm_init(): 创建一个二级页表。

void page_init(void): 将内存分页并初始化空闲页表。

`int page_alloc(struct Page **pp):` 分配一页内存并把值赋给pp。
`void page_free(struct Page *pp):` 释放一页内存。
`int pgdir_walk(Pde *pgdir, u_long va, int create, Pte **ppte):` 建立起二级页表结构后从页目录中找到va对应页表项的函数。
`int page_insert(Pde *pgdir, struct Page *pp, u_long va, u_int perm):` 将物理页pp映射到va。
`struct Page * page_lookup(Pde *pgdir, u_long va, Pte **ppte):` 找到虚拟地址va对应的物理页面。
`void page_decref(struct Page *pp):` 降低物理页面的引用次数，降到0后释放页面。
`void page_remove(Pde *pgdir, u_long va):` 释放虚拟地址va对应的页面。
`void tlb_invalidate(Pde *pgdir, u_long va):` 清空TLB中va对应的项。

物理地址或上标志位，虚拟地址加偏移量得到相应地址（在mos中keseg0虚拟地址和物理地址只差前三位置不置0,所以虚拟地址加和物理地址加偏移量是一样的），传进来传出去的都是虚拟地址

```
/OSLAB/gxemul -E testmips -C R3000 -M 64 gxemul/vmlinux
```