lab2上机

lab2-1

exam

题目背景

我们实现的 MOS 操作系统中，所有的物理页的可能状态有三种：使用中物理页、空闲物理页、已经被申请但未被使用的物理页。

> page_alloc 的时候只是申请了一个物理页，但是物理页没有使用，请仔细思考这三种状态物理页的判定方法。

说明：

1. 使用中的物理页：当前使用次数不为0的物理页，状态标记为1
2. 已经被申请但未使用的物理页：当前使用次数为0，但是已经被申请出去的物理页，状态标记为2
3. 空闲物理页：当前可以被申请的物理页，状态标记为3

任务1

在pmap.c中实现函数`int page_alloc2(struct Page **pp)`，并在 pmap.h 中添加该函数的声明。其功能与原有的 page_alloc 完全一样（你可以直接复制 page_alloc 的代码），唯一的区别在于，如果确实分配到了物理页面，该函数要输出分配到的物理页的信息。

输出格式：`printf("page number is %x, start from pa %x\n", ppn, pa);`，其中ppn为页号，pa为该页面的起始物理地址。

- pmap.c 中添加：

  （其中，`int page_alloc2(struct Page **pp)`添加了13 14 19行的三条语句）

```
int page_alloc2(struct Page **pp) {
    struct Page *ppage_temp;

    /* Step 1: Get a page from free memory. If fail, return the error
code.*/
    if (LIST_EMPTY(&page_free_list)) {
        return -E_NO_MEM;
    }
    ppage_temp = LIST_FIRST(&page_free_list);
    LIST_REMOVE(ppage_temp, pp_link);

    /* Step 2: Initialize this page.
     * Hint: use `bzero`. */
    u_long ppn = page2ppn(ppage_temp);
    u_long pa = page2pa(ppage_temp);

    bzero(page2kva(ppage_temp), BY2PG);
    *pp = ppage_temp;

    printf("page number is %x, start from pa %x\n", ppn, pa);
```

```
    return 0;
}
```

- pmap.h 中添加：

```
int page_alloc2(struct Page **pp);
```

任务2

在pmap.c中实现函数 `void get_page_status(int pa)` 并在pmap.h中添加该函数的声明。函数输入的是一个物理地址，请按格式输出该物理页的状态信息。

输出格式： `printf("times:%d, page status:%d\n", var1, var2);`

其中 var1 是统计该函数被调用的次数（首次从1开始），var2 是返回该物理地址对应的页面状态标记数字。
评测要求：请确保 page_init 初始化后 page_free_list 从表头到表尾物理页下标依次递减

```
int lab2_times = 0;
void get_page_status(int pa) {
    lab2_times++;
    struct Page *p = pa2page(pa);
    struct Page *tmp;
    int in_list = 0;
    LIST_FOREACH(tmp, &page_free_list, pp_link) {
        if (tmp == p) {
            in_list = 1;
        }
    }
    int status;
    if (p->pp_ref != 0) {
        status = 1;
    } else if (in_list == 0) {
        status = 2;
    } else {
        status = 3;
    }
    printf("times:%d, page status:%d\n", lab2_times, status);
}
```

任务3

本次课上测试会对课下测试进行加强测试，请大家在 pmap.h 中添加以下函数定义（请不要在 pmap.c 中添加这两个函数的实现，否则远端测评无法编译）：

1. `void test_queue();`
2. `void pm_check();`

Extra_2021

题目背景

常见的管理空闲内存的方法有两种：链表管理法与位图管理法。在 lab2 中，我们通过链表实现了对空闲内存的管理，下面的 extra 部分将要求大家实现用位图管理空闲内存。

用位图管理内存需要给每个分配单元赋予一个二进制数位，用来记录该分配单元是否闲置。

位图规格要求：

数位取值为 0 表示单元闲置，取值为 1 则表示已被占用。

用一个`unsigned int page_bitmap`数组管理内存，要求在该数组中，标号小的元素的低位表示页号小的页面。例如，0 号页面由 page_bitmap[0] 的第 0 位表示，63 号页面由 page_bitmap[1] 的第 31 位表示。当只有 0 号页面与 63 号页面被占用时，应该有：page_bitmap[0]=0x00000001，page_bitmap[1]=0x80000000

任务一

在 pmap.c 中添加如下的空闲页面位图定义：`unsigned int page_bitmap[NUM];`

其中 NUM 是一个你需要计算的数，要求这个数组可以恰好表示所有物理页面，不多不少。

修改`page_init()`除需要初始化位图外，需要添加输出：`printf("page bitmap size is %x\n", NUM);`

其中 NUM 为 page_bitmap 数组的元素个数。

1. 添加全局

```
#define MAPSIZE 512
unsigned int page_bitmap[MAPSIZE];
```

2. 修改`page_init()`

```
void page_init(void) {
    /* Step 1: Initialize page_free_list. */
    // LIST_INIT(&page_free_list);
    /* Hint: Use macro `LIST_INIT` defined in include/queue.h. */

    /* Step 2: Align `freemem` up to multiple of BY2PG. */
    freemem = ROUND(freemem, BY2PG);

    /* Step 3: Mark all memory blow `freemem` as used(set `pp_ref`
     * filed to 1) */
    int used_page = PADDR(freemem) / BY2PG;
    int i, j;
    int im, jm;
    for (i = 0; i < MAPSIZE; i++) {
        page_bitmap[i] = 0;
    }
    im = used_page / 32;
    jm = used_page % 32;
    for (i = 0; i < im; i++) {
        page_bitmap[i] = ~0;
    }
```

```
    for (j = 0; j < jm; j++) {
        page_bitmap[im] |= (1 << j);
    }
    printf("page bitmap size is %x\n", MAPSIZE);
}
```

任务二

修改`page_alloc(struct Page **pp)`，要求分配到的页面是空闲页面中页号最小的。

```
int page_alloc(struct Page **pp) {
    int pn;
    int i, j;
    unsigned int bits;
    for (i = 0; i < MAPSIZE; i++) {
        bits = page_bitmap[i];
        if (bits != ~0) {
            for (j = 0; j < 32; j++) {
                if ((bits & (1 << j)) == 0) {
                    bits |= (1 << j);
                    page_bitmap[i] = bits;
                    pn = i * 32 + j;
                    *pp = pages + pn;
                    return 0;
                }
            }
        }
    }
    return -E_NO_MEM;
}
```

任务三

修改`page_free(struct Page *pp)`

```
void page_free(struct Page *pp) {
    /* Step 1: If there's still virtual address refers to this page, do nothing.
*/

    if (pp->pp_ref > 0) {
        return;
    }

    int pn = page2ppn(pp);
    int i, j;
    /* Step 2: If the `pp_ref` reaches to 0, mark this page as free and return.
*/
    if (pp->pp_ref == 0) {
        i = pn / 32;
        j = pn % 32;
        page_bitmap[i] &= ~(1 << j);
        return;
    }
```

```
    /* If the value of `pp_ref` less than 0, some error must occurred before,
     * so PANIC !!! */
    panic("cgh:pp->pp_ref is less than zero\n");
}
```

三个函数修改后需要满足前述的位图规格要求。除页面组织形式外，其他要求与课下要求相同。

注意：请保证没有使用链表相关操作组织页面，评测时若发现使用链表组织页面将不予通过！

评测要求：为了正确评测，请在 pmap.h 中添加以下函数定义（请不要在 pmap.c 中添加这个函数的实现，否则远端测评无法编译）：

`void pm_check(void);`

Extra_2019

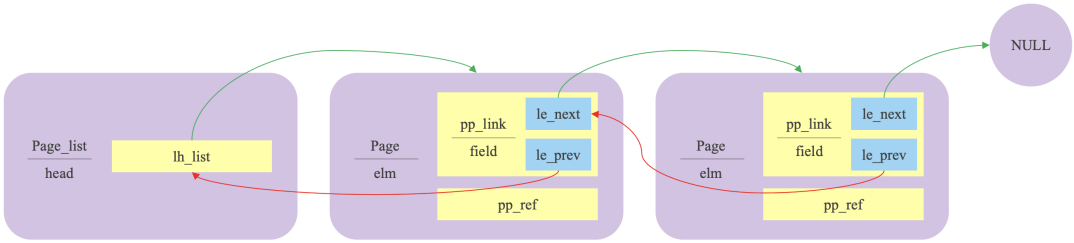
只贴出 login256 修改的地方。我才不会说是因为我也不知道他写的是啥的哼。

```
/* 去掉了前面的 static */
struct Page_list page_free_list; /* Free list of physical pages */

static void *alloc(u_int n, u_int align, int clear) {
    extern char end[];
    u_long alloced_mem;

    /* Initialize `freemem` if this is the first time. The first virtual address
that the
     * linker did *not* assign to any kernel code or global variables. */
    if (freemem == 0) {
        freemem = (u_long)(maxpa + ULIM);
    }

    /* Step 1: Round up `freemem` up to be aligned properly */
    /* Step 2: Save current value of `freemem` as allocated chunk. */
    alloced_mem = freemem - n;
    if (alloced_mem != ROUND(alloced_mem, align)) {
        alloced_mem = ROUND(alloced_mem, align) - align;
    }

    /* Step 3: Increase `freemem` to record allocation. */
    freemem = alloced_mem;

    /* Step 4: Clear allocated chunk if parameter `clear` is set. */
    if (clear) {
        bzero((void *) alloced_mem, n);
    }

    // We're out of memory, PANIC !!
    if (freemem <= end) {
        panic("out of memorty\n");
        return (void *) -E_NO_MEM;
    }
```

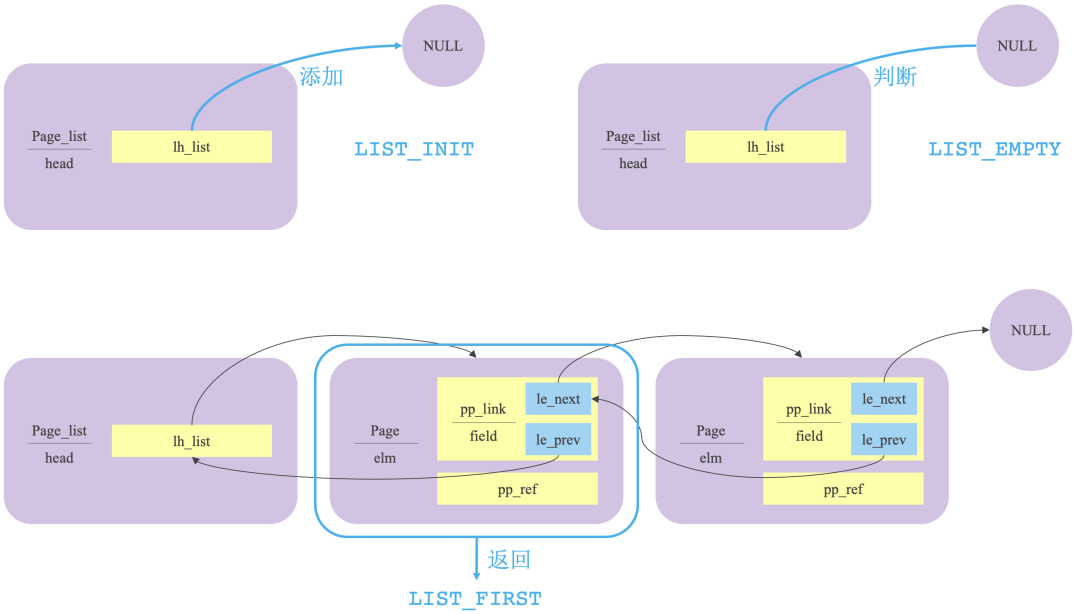
```
    /* Step 5: return allocated chunk. */
    return (void *) alloced_mem;
}

void page_init() {
    struct Page *now;
    struct Page *last;
    /* Step 1: Initialize page_free_list. */
    /* Hint: Use macro `LIST_INIT` defined in include/queue.h. */
    LIST_INIT(&page_free_list);
    /* Step 2: Align `freemem` up to multiple of BY2PG. */
    ROUND(freemem, BY2PG);

    /* Step 3: Mark all memory blow `freemem` as used(set `pp_ref`
     * filed to 1) */
    for (now = pages; page2kva(now) < freemem; now++) {
        now->pp_ref = 0;
        LIST_INSERT_HEAD(&page_free_list, now, pp_link);
    }

    /* Step 4: Mark the other memory as free. */
    for (now = &pages[PPN(PADDR(freemem))]; page2ppn(now) < npage; now++) {
        now->pp_ref = 1;
    }
}
```

喵呜~

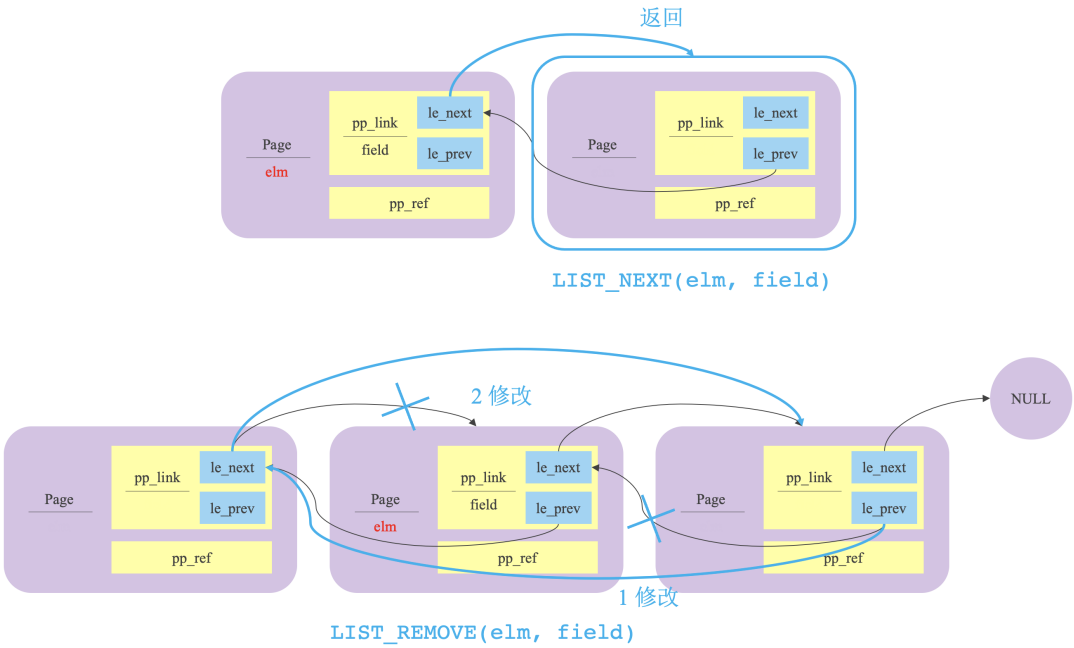
1. LIST 结构



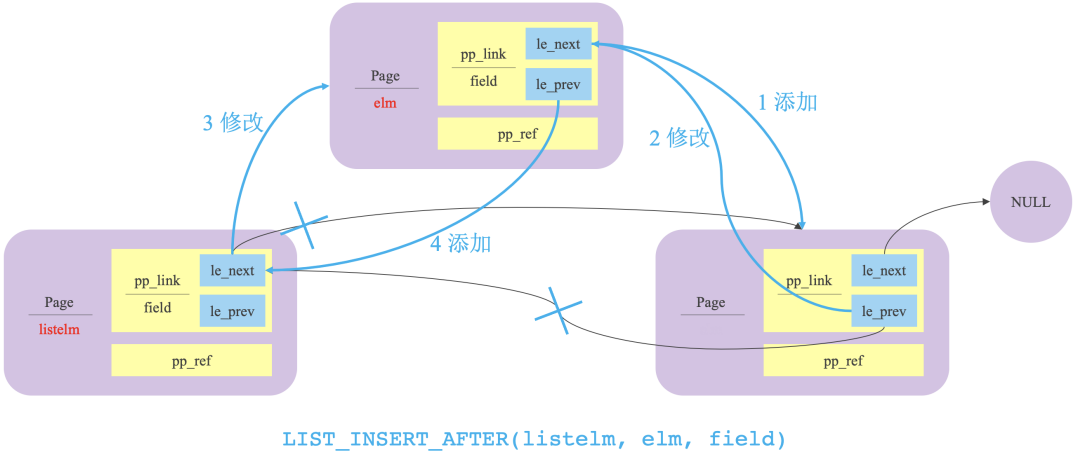
2. `LIST_INIT(head)`、`LIST_EMPTY(head)` 和 `LIST_FIRST(head)`



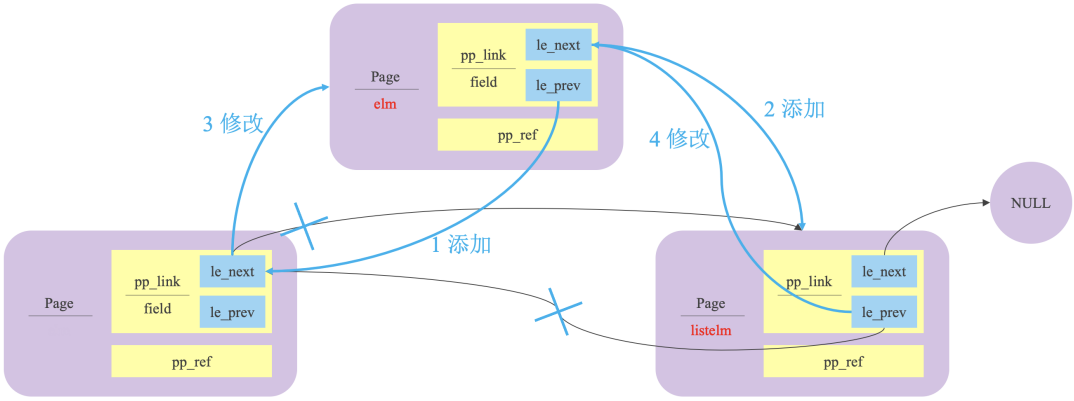
3. `LIST_NEXT(elm, field)`和`LIST_REMOVE(elm, field)`



4. `LIST_INSERT_AFTER(listelm, elm, field)`

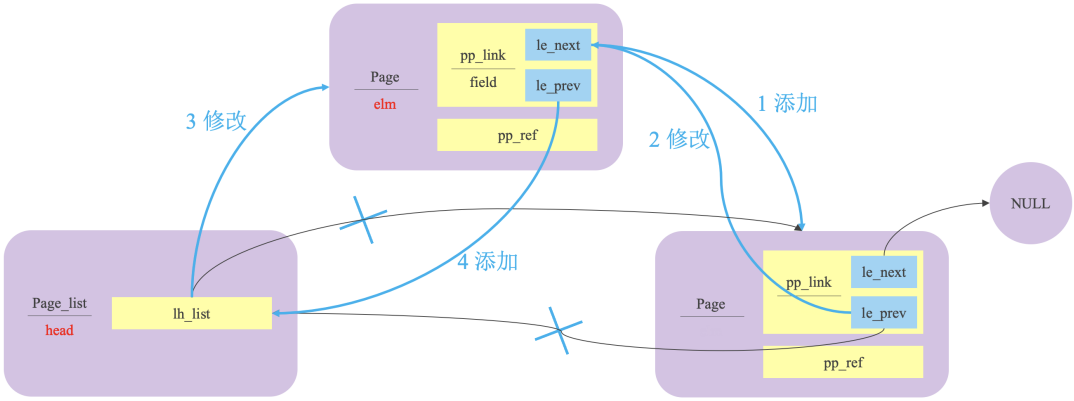


5. `LIST_INSERT_BEFORE(listelm, elm, field)`



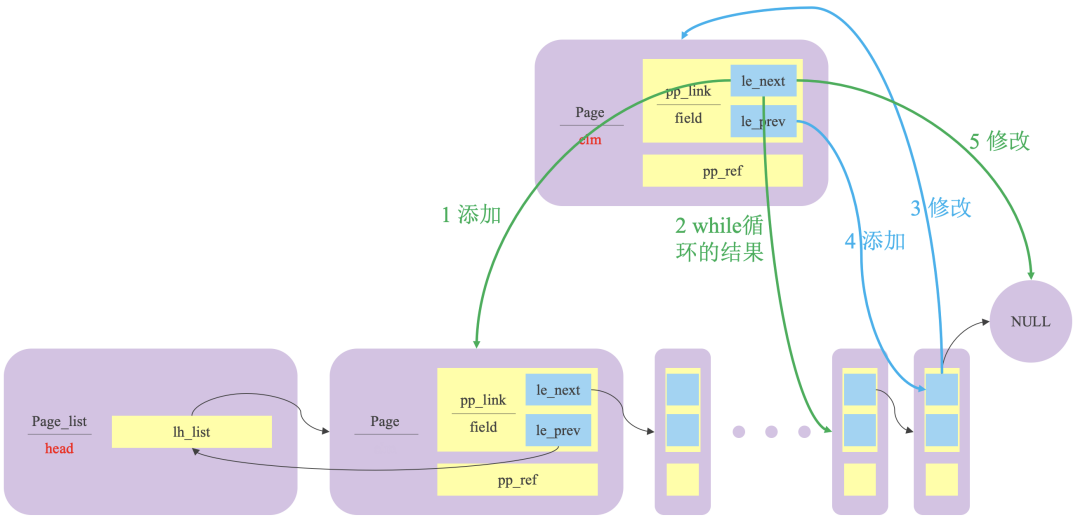
LIST_INSERT_BEFORE(listelm, elm, field)

6. `LIST_INSERT_HEAD(head, elm, field)`



LIST_INSERT_HEAD(head, elm, field)

7. `LIST_INSERT_TAIL(head, elm, field)`



LIST_INSERT_TAIL(head, elm, field)

lab2-2

exam_2021

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。

> 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 64 bit = 8 Byte，一个页表中有 $=2^9$项，三级页表，$total\_bit = 3 \times 9 + pgshift = 3 \times 9 + 12 = 39$。

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 0,1,2,3。

1. va为前述任务中的虚拟地址，func为0时，传入0。
2. 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。
3. 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。

```
u_long cal_page(int func, u_long va, int n, Pde *pgdir) {
    if (func == 0) {
        return (u_long) 39;
    }
    if (func == 1) {
        return (u_long)(va + ((va >> 12) << 2));
    }
    if (func == 2) {
        return (u_long)(((va >> 22) << 22) + ((n) << 12));
    }
    if (func == 3) {
```

```
        u_long *x;
        x = &pgdir[(va >> 22) & 0x3ff];
        *x = PADDR(pgdir) | PTE_V;
        return 0;
    }
    return 0;
}
```

exam_2019

只贴出 login256 的代码，由于 lab2-2-exam 他没有提供 result，不保证正确性。而且也不知道题目是什么，看看就行。

```
struct Page_list page_free_list; /* Free list of physical pages */

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;
}
```

Extra_2021

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

1. 在 pmap.c 文件中编写函数 `int count_page(Pde *pgdir, int *cnt)`
2. 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

1. 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。
2. 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

1. 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。
2. 一个物理页可能被进程的多个虚拟页映射。
3. 传入的 cnt 数组不一定全 0。

贴一个 84/100 的：

```
int count_page(Pde *pgdir, int *cnt) {
    int i, j;
    Pde *pgdir_entry;
    Pte *pgtable_entry;
    for (i = 0; i < npage; i++) {
        cnt[i] = 0;
    }
    cnt[PADDR(pgdir) >> 12]++;
    pgdir_entry = pgdir;
    for (i = 0; i < 1024; i++) {
        if (!(*pgdir_entry & PTE_V)) {
            continue;
        }
        cnt[((*pgdir_entry) >> 12)]++;
        pgtable_entry = KADDR(PTE_ADDR(*pgdir_entry));
        for (j = 0; j < 1024; j++) {
            if (!(*pgtable_entry & PTE_V)) {
                continue;
            }
            cnt[((*pgtable_entry) >> 12)]++;
            pgtable_entry++;
        }
        pgdir_entry++;
    }
    return npage;
}
```

Extra_2019

2019 年的 lab2-2-Extra 竟是 2021 年 lab2-2-exam 的削减版，令人感慨。

附录

1. 自映射相关计算

构建方法

1. 给定一个页表基址PT_{base}，该基址需4M对齐，即：

   $PT_{base} = ((PT_{base}) >> 22) << 22;$

   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？

   $PD_{base} = PT_{base} | (PT_{base}) >> 10$

3. 自映射目录表项$PDE_{self\text{-}mapping}$在哪里？

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