

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

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

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();
```

```
#define LIST_INSERT_TAIL(head, elm, field) do { \
    if (LIST_FIRST((head)) == NULL) { \
        \
        LIST_FIRST((head)) = (elm); \
        (elm)->field.le_prev = &LIST_FIRST((head)); \
    } else { \
        \
        for ((LIST_NEXT((elm), field)) = LIST_FIRST((head)); \
            \
            ((LIST_NEXT(LIST_NEXT((elm), field), field)) != \
            NULL); \
            \
            (LIST_NEXT((elm), field)) = \
            LIST_NEXT(LIST_NEXT((elm), field), field)); \
            LIST_NEXT(LIST_NEXT((elm), field), field) = (elm); \
            (elm)->field.le_prev = &LIST_NEXT(LIST_NEXT((elm), \
            field), field); \
    } \
    \
    LIST_NEXT((elm), field) = NULL; \
} while (0)

#define LIST_INSERT_AFTER(listelm, elm, field) do { \
    \
    LIST_NEXT((elm), field) = LIST_NEXT((listelm), field); \
    \
    if (LIST_NEXT((listelm), field) != NULL) \
    \

```

```

        LIST_NEXT((listelm), field)->field.le_prev =
&LIST_NEXT(elm, field); \
        LIST_NEXT((listelm), field) = (elm);
    \
    (elm)->field.le_prev = &LIST_NEXT((listelm), field);
    \
} while (0)

#define LIST_INSERT_AFTER(listelm, elm, field) do { \
    LIST_NEXT(elm, field) = LIST_NEXT((listelm), field); \
    if (LIST_NEXT((listelm), field)) { \
        LIST_NEXT((listelm), field)->field.le_prev =
&LIST_NEXT(elm, field); \
    } \
    LIST_NEXT((listelm), field) = (elm); \
    (elm)->field.le_prev = &LIST_NEXT((listelm), field); \
}while (0)

#define LIST_INSERT_TAIL(head, elm, field) do { \
    if (LIST_FIRST((head)) != NULL) { \
        LIST_NEXT(elm, field) = LIST_FIRST((head)); \
        while (LIST_NEXT(LIST_NEXT(elm), field), field) !=
NULL) { \
            LIST_NEXT(elm, field) = LIST_NEXT(LIST_NEXT(elm),
field), field); \
        } \
        LIST_NEXT(LIST_NEXT(elm), field), field) = (elm); \
        (elm)->field.le_prev = &LIST_NEXT(LIST_NEXT(elm),
field), field); \
        LIST_NEXT(elm, field) = NULL; \
    } else { \
        LIST_INSERT_HEAD((head), (elm), field); \
    } \
} while (0)

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 = PADDD(freemem)/BY2PG;
    int i;
    for (i = 0; i < used_page; i++) {
        pages[i].pp_ref = 1;
    }
}

```

```

/* Step 4: Mark the other memory as free. */
for (i = used_page; i < npage; i++) {
    pages[i].pp_ref = 0;
    LIST_INSERT_HEAD(&page_free_list, &pages[i], pp_link);
}
}

int
page_alloc(struct Page **pp)
{
    struct Page *ppage_temp;

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

    /* Step 2: Initialize this page.
     * Hint: use `bzero`. */
    LIST_REMOVE(ppage_temp, pp_link);
    u_long temp = page2kva(ppage_temp);
    bzero((void *)temp, BY2PG);
    *pp = ppage_temp;
    return 0;
}

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;
    }

    /* Step 2: If the `pp_ref` reaches to 0, mark this page as free and return.
     */
    if(pp->pp_ref == 0) {
        LIST_INSERT_HEAD(&page_free_list, pp, pp_link);
        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");
}

```

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

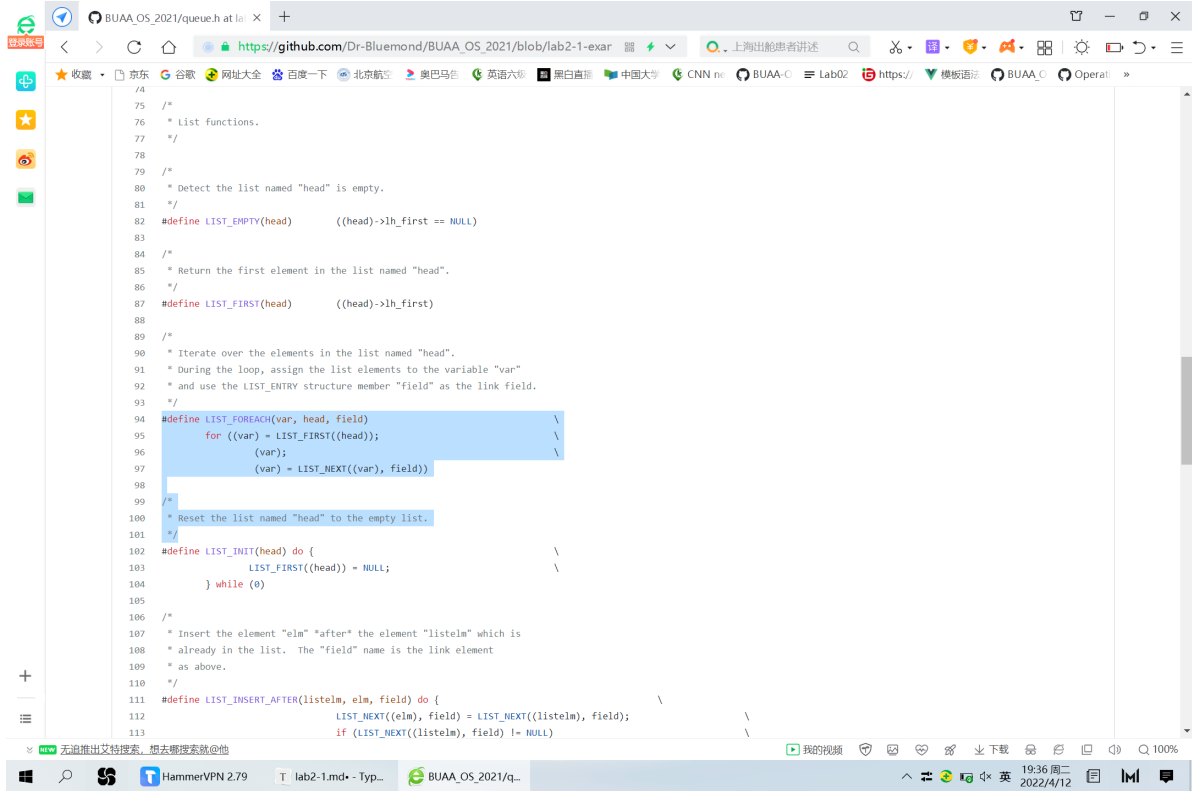
page2pa(struct Page *pp) 得到页pp的实地址

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

```
Operating-System-BUAA-202 BUAA_OS_2021/pmap.c at lab2-1-exam/mr
https://github.com/Dr-Bluemond/BUAA_OS_2021/blob/lab2-1-exam/mr
270
271 int
272 page_alloc2(struct Page **pp)
273 {
274     struct Page *ppage_temp;
275
276     /* Step 1: Get a page from free memory. If fails, return the error code.*/
277     if ((ppage_temp = LIST_FIRST(&page_free_list)) == NULL) {
278         return -E_NO_MEM;
279     }
280
281     /* Step 2: Initialize this page.
282      * Hint: use `bzero`. */
283     LIST_REMOVE(ppage_temp, pp_link);
284     u_long temp = page2kva(ppage_temp);
285
286     u_long ppn = page2ppn(ppage_temp);
287     u_long pa = page2pa(ppage_temp);
288
289     bzero((void *)temp, BY2PG);
290     *pp = ppage_temp;
291     printf("page number is %Xx, start from pa %Xx\n", ppn, pa);
292
293     return 0;
294 }
295
296 }
297
298 int lab2_times = 0;
299
300 void get_page_status(int pa) {
301     lab2_times++;
302     struct Page *p = pa2page(pa);
303     struct Page *tmp;
304     int in_list = 0;
305     LIST_FOREACH(tmp, &page_free_list, pp_link) {
306         if (tmp == p) {
307             in_list = 1;
308         }
309     }
310
311     int status;
312     if (p->pp_ref != 0) {
313         status = 1;
314     } else if (in_list == 0) {
315         status = 2;
316     } else {
317         status = 3;
318     }
319     printf("times:%d, page status:%d\n", lab2_times, status);
320 }
321
322 /*Overview:
323  Release a page, mark it as free if it's `pp_ref` reaches 0.
324  Hint:
325  When to free a page, just insert it to the page_free_list.*/
326 void
327 page_free(struct Page *pp)
328 {
329     /* Step 1: If there's still virtual address refers to this page, do nothing. */
330
331     if(pp->pp_ref > 0) {
332         return;
333     }
334
335     /* Step 2: If the `pp_ref` reaches to 0, mark this page as free and return. */
336     if(pp->pp_ref == 0) {
337         LIST_INSERT_HEAD(&page_free_list, pp, pp_link);
338         return;
339     }
340
341     /* If the value of `pp_ref` less than 0, some error must occurred before,
342      * so PANIC !!! */
343     panic("cgh:pp->pp_ref is less than zero\n");
344 }
345
346 }
```

```
Operating-System-BUAA-202 BUAA_OS_2021/pmap.c at lab2-1-exam/mr
https://github.com/Dr-Bluemond/BUAA_OS_2021/blob/lab2-1-exam/mr
308
309 }
310
311 int status;
312 if (p->pp_ref != 0) {
313     status = 1;
314 } else if (in_list == 0) {
315     status = 2;
316 } else {
317     status = 3;
318 }
319 printf("times:%d, page status:%d\n", lab2_times, status);
320 }
321
322 /*Overview:
323  Release a page, mark it as free if it's `pp_ref` reaches 0.
324  Hint:
325  When to free a page, just insert it to the page_free_list.*/
326 void
327 page_free(struct Page *pp)
328 {
329     /* Step 1: If there's still virtual address refers to this page, do nothing. */
330
331     if(pp->pp_ref > 0) {
332         return;
333     }
334
335     /* Step 2: If the `pp_ref` reaches to 0, mark this page as free and return. */
336     if(pp->pp_ref == 0) {
337         LIST_INSERT_HEAD(&page_free_list, pp, pp_link);
338         return;
339     }
340
341     /* If the value of `pp_ref` less than 0, some error must occurred before,
342      * so PANIC !!! */
343     panic("cgh:pp->pp_ref is less than zero\n");
344 }
345
346 }
```



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。

Note: `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：

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

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

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

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

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）：

```

void test_queue();
void pm_check();


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

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

    /* Step 2: Initialize this page.
     * Hint: use `bzero`. */
    LIST_REMOVE(ppage_temp, pp_link);
    u_long temp = page2kva(ppage_temp);

    u_long ppn = page2ppn(ppage_temp);
    u_long pa = page2pa(ppage_temp);

    bzero((void *)temp, BY2PG);
    *pp = ppage_temp;
    printf("page number is %x, start from pa %x\n", ppn, pa);

    return 0;
}

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);
}

```

EXTRA



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

题目要求：

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

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

任务三

修改page_free(struct Page *pp)

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）：

```
pm_check(void);
```

```
#define MAPSIZE 512
unsigned int page_bitmap[MAPSIZE];
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[i] |= (1 << j);
    }
    printf("page bitmap size is %x\n", MAPSIZE);
}

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;
}

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");
}

```