Lab9 Report

> 12011702 张镇涛

Q1

宏展开

```
52
53 // convert list entry to page
54 #define le2page(le, member)
55     to_struct((le), struct Page, member)
56
```

在kern/mm/memlayout.h中寻找le2page的宏定义。

接着在lib/def.h中继续寻找to_struct的定义：

```
9 /* *
0  * to_struct - get the struct from a ptr
1  * @ptr:    a struct pointer of member
2  * @type:   the type of the struct this is embedded in
3  * @member: the name of the member within the struct
4  * */
5 #define to_struct(ptr, type, member)
6     ((type *)((char *)(ptr) - offsetof(type, member)))
7
```

发现其调用了offsetof()方法，继续寻找offsetof()的定义

```
5 /* Return the offset of 'member' relative to the beginning of a struct type */
6 #define offsetof(type, member)                                      \
7     ((size_t)(&((type *)0)->member))
8
```

回到default_pmm.c第85行代码

```
struct Page* page = le2page(le, page_link);
```

做逐步展开，得到以下：

```
to_struct((le), struct Page, page_link)
((struct Page*)((char*)((le))-offsetof(struct Page, page_link)))
```

最后展开得到：

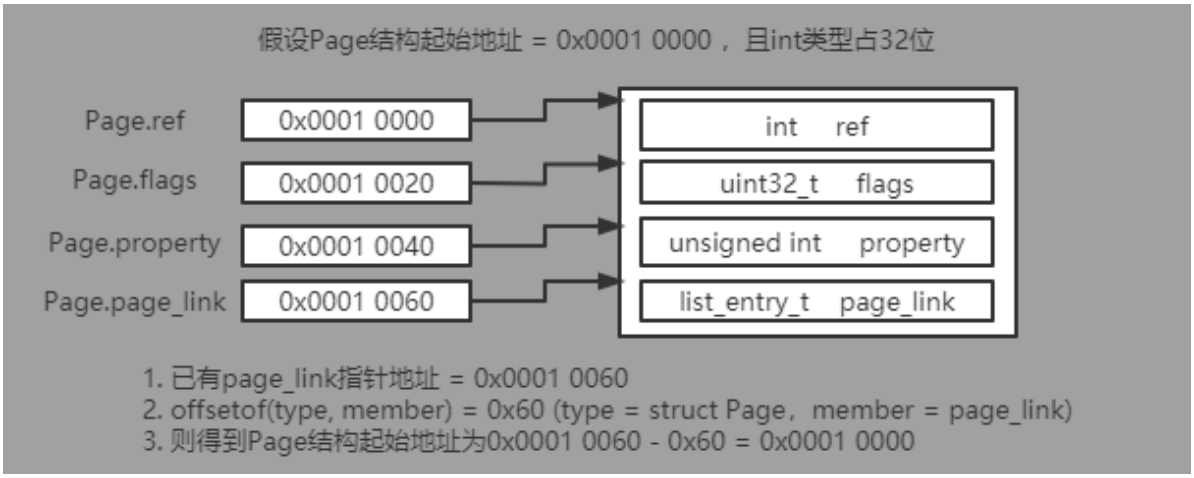
```
((struct Page*)((char*)((le))-((size_t)(&((struct Page *)0)->page_link))
```

工作原理

首先需要查看Page的struct结构:

```
/* *
 * struct Page - Page descriptor structures. Each Page describes one
 * physical page. In kern/mm/pmm.h, you can find lots of useful functions
 * that convert Page to other data types, such as physical address.
 * */
struct Page {
    int ref;                        // page frame's reference counter
    uint64_t flags;                 // array of flags that describe the status of the page frame
    unsigned int property;          // the num of free block, used in first fit pm manager
    list_entry_t page_link;         // free list link
};
```

首先通过结构体中的定义得到page_link的指针地址，然后获得page_link到结构起始地址的相对偏移（offsetof函数），这样就获取了page_link节点所属Page结构的首地址，最后强制转换为对应的Page指针。



图片来源：https://www.bbsmax.com/A/E35pW22gJv/

Q2

default_alloc_pages

功能

用于分配物理内存页，函数参数为一个正整数n，该函数为其分配n个物理大小页面的连续物理内存空间，并返回指向最前面的最低位物理内存页的Page指针。

如果分配时发生错误或者剩余空闲空间不足，则返回NULL代表分配失败

实现方式

分析其代码，首先检验参数合法性。

```
assert(n > 0);
if (n > nr_free) {
    return NULL;
}
```

如果正整数n比剩余空闲空间大，则直接返回NULL

```
        struct Page *page = NULL;
        list_entry_t *le = &free_list;
        while ((le = list_next(le)) != &free_list) {
            struct Page *p = le2page(le, page_link);
            if (p->property >= n) {
                page = p;
                break;
            }
        }
        if (page != NULL) {
```

遍历空闲链表，直到发现一个满足要求的，空闲页数大于等于n的空闲块

```
    if (page != NULL) {
        list_entry_t* prev = list_prev(&(page->page_link));
        list_del(&(page->page_link));
        if (page->property > n) {
            struct Page *p = page + n;
            p->property = page->property - n;
            SetPageProperty(p);
            list_add(prev, &(p->page_link));
        }
        nr_free -= n;
        ClearPageProperty(page);
    }
```

找到了以后就进行物理内存页的分配。

将当前page从空闲链表中移除。如果空闲块的大小不是正合适(page->property != n)，则按照指针偏移，找到按序后面第n个Page结构p；更新p的空闲块个数，并且按对应的物理地址顺序，将p加入到空闲链表中对应的位置。

最后返回相应分配好的物理内存的指针，以Page*的方式返回。

default_free_pages

功能

接受一个Page指针和正整数n，用于释放物理内存页，释放掉自base起始的连续n个物理页,n必须为正整数。

实现方式

```
    assert(n > 0);
    struct Page *p = base;
    for (; p != base + n; p ++) {
        assert(!PageReserved(p) && !PageProperty(p));
        p->flags = 0;
        set_page_ref(p, 0);
    }
    base->property = n;
    SetPageProperty(base);
    nr_free += n;
```

首先依然是检验参数合法性，然后遍历从地址最前page开始的n个page，并设置相关属性设置为空闲，然后更新相关属性。

```
if (list_empty(&free_list)) {
    list_add(&free_list, &(base->page_link));
} else {
    list_entry_t* le = &free_list;
    while ((le = list_next(le)) != &free_list) {
        struct Page* page = le2page(le, page_link);
        if (base < page) {
            list_add_before(le, &(base->page_link));
            break;
        } else if (list_next(le) == &free_list) {
            list_add(le, &(base->page_link));
        }
    }
}
```

这一段代码的作用是找到第一个比base大的page，然后插在它前面。这样就维持了空闲块的链表按照空闲块的地址有序排列。