Week6 Report

姓名：Yitong WANG(王奕童) 11910104@mail.sustech.edu.cn

学号：11910104

实验课时段：周五5-6节

实验课教师：Yun SHEN(沈昀) sheny@mail.sustech.edu.cn

实验课SA:

- Yining TANG(汤怡宁) 11811237@mail.sustech.edu.cn
- Yushan WANG(王宇杉) 11813002@mail.sustech.edu.cn

Q1 `le2page` 宏展开与工作原理

Q1-1 宏展开

先在 `kern/mm/memlayout.h` 中寻找 `le2page` 的相关宏定义，如下：

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

然后在 `libs/defs.h` 中继续寻找宏定义，如下：

```
65  /* Return the offset of 'member' relative to the beginning of a struct type */
66  #define offsetof(type, member)                                      \
67      ((size_t)(&((type *)0)->member))
68
69  /* *
70   * to_struct - get the struct from a ptr
71   * @ptr:    a struct pointer of member
72   * @type:   the type of the struct this is embedded in
73   * @member: the name of the member within the struct
74   * */
75  #define to_struct(ptr, type, member)                               \
76      ((type *)((char *)(ptr) - offsetof(type, member)))
77
```

现考虑85行的对应代码：

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

做逐步展开，如下：

```
le2page(le, page_link)

to_struct((le), struct Page, page_link)

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

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

Q1-2 工作原理

先查看 `Page` 结构体的定义，位置在 `kern/mm/memlayout.h` ：

```
35  struct Page {
36      int ref;                        // page frame's reference counter
37      uint64_t flags;                 // array of flags that describe the status of the page frame
38      unsigned int property;          // the num of free block, used in first fit pm manager
39      list_entry_t page_link;         // free list link
40  };
```

这里的工作原理是首先通过结构体中的定义得到 `page_link` 中的偏移位置（注意这里offset中的零指针，并非运行时访问，而是编译期就会自动计算替代），
然后就可以通过相减得到 `page_link` 节点所在的 `Page` 结构的首地址，最后做强制类型转换以获得 `struct Page*` 。

引用以下参考链接中的一句话：offset宏的构造，是C语言中通过结构体中某一属性地址访问所属其所在结构体的一种巧妙实现。

参考链接：https://www.bbsmax.com/A/E35pW22gJv/

Q2 `default_alloc_pages` 和 `default_free_pages` 的功能与实现方式

功能

- `default_alloc_pages` ： 分配物理内存页，接受一个正整数n，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

- `default_free_pages`：释放物理内存页，接受一个Page的基址base和正整数n，释放掉自base起始的连续n个物理内存页。并按照空闲块的地址做有序的空闲链表排列。

参考链接：https://www.bbsmax.com/A/E35pW22gJv/

实现方式

以下将参照代码进行实现方式的说明：

default_alloc_pages

```
98      assert(n > 0);
99      if (n > nr_free) {
100         return NULL;
101     }
```

检查参数合法性，参数n必须是正数。m不能比剩余的空闲空间nr_free大，否则分配失败。

```
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;
    }
}
```

遍历空闲链表，寻求第一个空闲位置大小大于等于n的空闲块（原理类似于first-fit）。

```
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);
}
return page;
```

如果不是NULL，那么就说明找到了；找到了以后就进行物理内存页的分配即可。
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

```
    assert(n > 0);
```

检查参数合法性：传入的n必须是正数

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

遍历这n个连续的内存页，将相关的内存属性设置为空闲。

```
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));
        }
    }
}
```

这里是需要维持空闲块的链表按照空闲块的地址有序排列。需要寻求到第一个比base大的page，然后插在它前面。这样的有序排列的形成可以有效降低合并空闲块的时间开销。