# CS302: Assignment5 Report

Name：陆荻芸 SID：12011537

## 1 Answers of Question 1

There are 2 possible places of free blocks that can be merged: before or after the base block. So check them separately if there are free blocks. If the free block is right before or after base with n offset, it means that they are continuous and we should reset their properties and let base points to the front of this larger block. Also remove and clear the merged page.

```
//----------------------合并空闲块-------------------

// check after block
list_entry_t* le = list_next(&(base->page_link));
if (le != &free_list) {
    p = le2page(le, page_link);
    // if they are continuous, update properties to merge
    if (base + base->property == p) {
        base->property += p->property;
        ClearPageProperty(p);
        list_del(&(p->page_link));
    }
}
// check previous block
le = list_prev(&(base->page_link));
if (le != &free_list) {
    p = le2page(le, page_link);
    // if p is right in the front of base, merge
    if (p + p->property == base) {
        p->property += base->property;
        ClearPageProperty(base);
        list_del(&(base->page_link));
        base = p;
    }
}
```

(a) Modified Code

```
MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...
memory management: default_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x000000087fffff].
check_alloc_page() succeeded!
QEMU: Terminated
(base) ldy12011537@ludiyun-ROG:~/Desktop/OSlab&As/As5/lab9$
```

(b) QEMU Result

# 2 Answer of Question 2

The only difference between first fit and best fit algorithm is their allocated page. In best fit algorithm we should maintain a `cur_proper` to record the most approximate property of the page in the free list till we go through the whole list. Then do the page allocation afterwards.

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

(c) Modified Code

```
MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...
memory management: best_fit_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x0000000087ffffff].
check_alloc_page() succeeded!
QEMU: Terminated
(base) ldy12011537@ludiyun-ROG:~/Desktop/OSlab&As/As5/lab9$
```

(d) QEMU Result

Other parts of codes are shown below.

```
static void
best_fit_init(void)
{
    // TODO
    list_init(&free_list);
    nr_free = 0;
}

static void
best_fit_init_memmap(struct Page *base, size_t n)
{
    // TODO
    assert(n > 0);
    struct Page *p = base;
    for (; p != base + n; p ++) {
        assert(PageReserved(p));
        p->flags = p->property = 0;
        set_page_ref(p, 0);
    }
    base->property = n;
    SetPageProperty(base);
    nr_free += n;
    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));
            }
        }
    }
```

(e) init and

```
static void
best_fit_free_pages(struct Page *base, size_t n)
{
    // TODO
    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;

    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));
            }
```

(f) free pages 1

```
    list_entry_t *le = list_next(&(base->page_link));
    if (le != &free_list)
    {
        p = le2page(le, page_link);
        if (p == base + base->property)
        {
            base->property += p->property;
            ClearPageProperty(p);
            list_del(&(p->page_link));
        }
    }
    le = list_prev(&(base->page_link));
    if (le != &free_list)
    {
        p = le2page(le, page_link);
        if (p + p->property == base)
        {
            p->property += base->property;
            ClearPageProperty(base);
            list_del(&(base->page_link));
            base = p;
        }
    }
}
```

(g) free pages 2