实验报告

Lab2

姓名：朱瑞媛

学号：14307130373

注意事项：

1.请在每个 exercise 之后简要叙述实验原理，详细描述实验过程。
2.请将你认为的关键步骤附上必要的截图。
3.有需要写代码的实验，必须配有代码、注释以及对代码功能的说明。
4.你还可以列举包括但不局限于以下方面：实验过程中碰到的问题、你是如何解决的、实验之后你还留有哪些疑问和感想。

5.请在截止日期前将代码和报告上传到 ftp 的指定目录下，文件名为 os_lab1_学号.zip，该压缩文件中应包含实验报告和代码，其中实验报告格式为 pdf，置于压缩文件的根目录。
6.如果实验附有 question，请在对应 exercise 后作答，这是实验报告评分的重要部分。
7.Challenge 为加分选作题。每个 lab 可能有多个 challenge，我们会根据完成情况以及难度适当加分，这部分的实验过程描述应该比 exercise 更加详细。

Exercise 1. In the file kern/pmap. c, you must implement code for the following functions (probably in the order given).

boot_alloc()
mem_init() (only up to the call to check_page_free_list(1))
page_init()
page_alloc()
page_free()

check_page_free_list() and check_page_alloc() test your physical page allocator. You should boot JOS and see whether check_page_alloc() reports success. Fix your code so that it passes. You may find it helpful to add your own assert()s to verify that your assumptions are correct.

上述函数已经完成。下面是详细说明。

boot_alloc():

```
83 // before the page_free_list list has been set up.
84 static void *
85 boot_alloc(uint32_t n)
86 {
87     static char *nextfree;  // virtual address of next byte of free memory
88     char *result;
89
90     // Initialize nextfree if this is the first time.
91     // 'end' is a magic symbol automatically generated by the linker,
92     // which points to the end of the kernel's bss segment:
93     // the first virtual address that the linker did *not* assign
94     // to any kernel code or global variables.
95 +---  4 lines: if (!nextfree) {----------------------------------------------------------------------------------
99
100    // Allocate a chunk large enough to hold 'n' bytes, then update
101    // nextfree.  Make sure nextfree is kept aligned
102    // to a multiple of PGSIZE.
103    //
104    // LAB 2: Your code here.
105    if (n > 0)
106    {
107        result = nextfree;
108        nextfree = ROUNDUP((char *)(nextfree + n), PGSIZE);
109        if ((int)nextfree - KERNBASE > (npages * PGSIZE))
110            panic("We're out of memory.\n");
111        return result;
112    }
113    else
114        return nextfree;
115    return NULL;
116 }
```

这个函数只在 JOS 初始设置虚拟内存系统时使用。
n > 0 时分配可以容纳 n bytes 的 pages. 返回分配到的虚拟地址，将 nextfree 指向新的下一个空闲内存地址。如果超过最大内存，panic 错误信息。

mem_init():

```
152
153     //////////////////////////////////////////////////////////////////////
154     // Allocate an array of npages 'struct PageInfo's and store it in 'pages'.
155     // The kernel uses this array to keep track of physical pages: for
156     // each physical page, there is a corresponding struct PageInfo in this
157     // array.  'npages' is the number of physical pages in memory.  Use memset
158     // to initialize all fields of each struct PageInfo to 0.
159     // Your code goes here:
160     pages = (struct PageInfo *)boot_alloc(sizeof(struct PageInfo) * npages);
161     memset(pages, 0, sizeof(struct PageInfo) * npages);
```

分配一个记录 npages 页空闲情况信息的数组 pages，将其置 0.

```
163     //////////////////////////////////////////////////////////////////////
164     // Now that we've allocated the initial kernel data structures, we set
165     // up the list of free physical pages. Once we've done so, all further
166     // memory management will go through the page_* functions. In
167     // particular, we can now map memory using boot_map_region
168     // or page_insert
169     page_init();
170
171     check_page_free_list(1);
172     check_page_alloc();
173     check_page();
174
```

在 mem_init 函数中调用 page_init 函数，check_page_alloc 函数。
check_page_alloc 函数调用 page_alloc 函数及 page_free 函数。

page_init():

```
261     // Change the code to reflect this.
262     // NB: DO NOT actually touch the physical memory corresponding to
263     // free pages!
264     size_t i;
265     /*for (i = 0; i < npages; i++) {
266         pages[i].pp_ref = 0;
267         pages[i].pp_link = page_free_list;
268         page_free_list = &pages[i];*/
269     for (i = 0; i < npages_basemem; i++)
270     {
271         if (i == 0)
272             pages[i].pp_ref = 1;
273         else
274         {
275             pages[i].pp_ref = 0;
276             pages[i].pp_link = page_free_list;
277             page_free_list = &pages[i];
278         }
279     }
280     int occup = (int)ROUNDUP(((char *)pages) + npages * (sizeof(struct PageInfo)) - KERNBASE, PGSIZE)/PGSIZE;
281     for (i = npages_basemem; i < npages; i++)
282     {
283         if (i < occup)
284             pages[i].pp_ref = 1;
285         else
286         {
287             pages[i].pp_ref = 0;
288             pages[i].pp_link = page_free_list;
289             page_free_list = &pages[i];
290         }
291     }
292 }
```

判断哪些 pages 空闲，并将他们放到 page_free_list 链表中。
物理内存被分为 3 部分。
第一部分 base memory,第 0 页被占用，其它空闲。
第二部分 IO hole,不能使用。
第三部分 extended memory,pages 数组存放的部分被占用，其余空闲。
将空闲页加入 page_free_list 链表并对其信息进行设置。对于已使用的页的信息也需要进行设置。

page_alloc():

```
// Allocates a physical page.  If (alloc_flags & ALLOC_ZERO), fills the entire
// returned physical page with '\0' bytes.  Does NOT increment the reference
// count of the page - the caller must do these if necessary (either explicitly
// or via page_insert).
//
// Be sure to set the pp_link field of the allocated page to NULL so
// page_free can check for double-free bugs.
//
// Returns NULL if out of free memory.
//
// Hint: use page2kva and memset
struct PageInfo *
page_alloc(int alloc_flags)
{
    // Fill this function in
    if (page_free_list != NULL)
    {
        struct PageInfo *result;
        result = page_free_list;
        page_free_list = page_free_list -> pp_link;
        result -> pp_link = NULL;
        result -> pp_ref = 1;
        if (alloc_flags & ALLOC_ZERO)
            memset(page2kva(result), 0, PGSIZE);
        return result;
    }
    else
        return NULL;
}
```

分配物理页，并返回被分配的页的 PageInfo 类型的信息。
在 page_free_list 中分配出一个空闲页。if(alloc_flags & ALLOC_ZERO),将此页中所有 byte 置 0.
如果没有空闲页，返回 NULL.

page_free():

```
//
// Return a page to the free list.
// (This function should only be called when pp->pp_ref reaches 0.)
//
void
page_free(struct PageInfo *pp)
{
    // Fill this function in
    // Hint: You may want to panic if pp->pp_ref is nonzero or
    // pp->pp_link is not NULL.
    if (pp -> pp_ref == 0)
        panic("pp->pp_ref is zero.\n");
    if (pp -> pp_link != NULL)
        panic("pp->pp_link is not NULL.\n");
    pp -> pp_ref = 0;
    pp -> pp_link = page_free_list;
    page_free_list = pp;
}
```

判断等待释放的页是否为已使用的页。如果不是，panic 错误信息。
将释放后的空闲页加入 page_free_list.

实验感想：
万事开头难，经过 lab1 之后再做这个 lab 感觉上手容易许多。kern/pmap.c 中大量的注释以及 inc/memlayout.h 中的内存分配结构图使得理解变的更加容易。当然各个变量，函数头文件的层层调用使得寻找一些函数，变量的意义难以发现及理解，但是在寻找它们的过程中也熟悉了许多原本不太了解的 linux 终端指令。每做一个新的 lab,都能获得意想不到的新收获。