L1：pmm.c 实验报告

221220048 潘尧

一、 实验过程以及思路介绍

采用了类似于 slab 的方法，为大内存分配和小内存分配分别建立了 fast path 和 slow path 这两种路径选择，以求得内存分配和时间效率的相对协调。

我本来想一下子去实现这个功能的，结果写了两天发现不行，还是得一步一步的先从一个简单的实现做起，于是乎有了第一版 3/6 的一把大锁保平安的做法，同时也是我后续的为大内存分配和 cpu 分配页的内存分配方式，即从后往前分配，这样可以尽可能的减少 freenode 的产生（PS：我对于因为对齐而没有被利用的内存采用的做法是一并划入分配给它的内存中而不是新建一个 freenode 节点。这样子的做法显而易见的好处是可以减少 freenode 节点的加入和合并，后续我还发现了一个好处，就是：对于大内存分配这个做法似乎浪费了内存，但是对于小内存分配这个页面的时候，却能在一次分配中分配出更大的页面，这样就使得分配次数减少，碎片化就相对缓和了，而且也符合 jyy 说的那种大内存分配较为罕见的实际 workload，同时直接用于内存分配的地方变多了。）

后续在一把大锁保平安的基础之上我又添加了 cpu 分配页，然后页内分配的过程，这个过程想是挺好想的，但是有一点还是有点讲究的，就是一次页面分配多大的问题，分配小了，就需要多次分配多次动用 kernel 大锁，分配大了在 freenode 的时候就更不容易去把一个页面清空，然后归还这个页，导致压力测试失败。

这里就不得不提一嘴我在页内分配的时候的实现，我采用了一种似乎是

更加容易地做法，就是我一个页再分配完了之后一定要在这个页的内存全部都清空的时候才会释放，这就是上一段我为什么说分配的页面也不能太大的原因，这种做法我觉得是有一定的理论基础的，因为实际上我们进行内存释放的时候很多时候确实是一整块一整块的释放，或者说是，时间上相近申请的内存释放的时间也差不多，这样的话，理论上存在的多次申请然后只释放一个故意卡的压力测试失败的情况就很少见了，其实实在不行还能加上一个时间判定（cpu）。（补充一下：我的页内的内存没有头节点哦，这样就省下了因为对齐而导致的小内存分配空间浪费问题，他是一个固定的类似于 buddy 思路的做法，就是一个页他只有一种大小的内存不会说的导致这个页出现不对其的情况的）

```
typedef struct cpu_pagelist{
    lock_t cpu_lock;
    struct cpu_pageheader* page_header;
    struct cpu_pageheader* page_header32;
    struct cpu_pageheader* page_header64;
    struct cpu_pageheader* page_header128;
    struct cpu_pageheader* other_header;
}cpu_pagelist;
```

最后一个可能是有点想法的设计就是把那个 freenode 节点和头节点大小改为一样的，这样话在全局 free 的过程中，我就只需要把 occupied node 这个节点情空然后赋上对应的 freenode 节点的指针值。

在 cpu 持有的页内的指针的释放，实际上也会有一个读 occupied node 的过程，但是他得 magic 码不正确，这样就会识别出这个东西是在页内而不是单独申请的大空间。

二、 印象深刻的 bug

很遗憾，很惭愧，本人遇到的耗时最长的 bug 不是并发，也不是地址内存

访问越界、栈溢出等高级问题，而是一个很离谱，我自己看了都想笑的却花了一整天才解决的 bug。（一定要加括号啊啊啊啊啊!!!!!!）

Ps:这个图应该一看就懂了

```
    cpu_page->space_left = cpu_page->size / 4 * 1024 - 1;
```

对于这个玩意儿，我的评价是，代码的可读性确实要注意，如果我一开始就去#define page_size 4 * 1024，就不会出现这个问题了（悲）