# L1 实验报告

## 实验进度

全部完成

## 1. 特别的设计

### 1.1. slow path: buddy system

buddy system 概述：

我采用 buddy system 作为 slow path 的分配器。这个系统采用类似线段树的方式来划分内存上不同的区间，主要依靠为每一种大小的内存维护一个表示空闲区块的链表来记录当前内存的状态，通过内存的合并与拆分来灵活地分配内存。buddy system 可以有效减少 external fragmentation，并能一定程度上控制 internal fragmentation，同时，它可以方便地实现内存的对齐，满足实验的要求。

其实我觉得 buddy system 并不复杂，它的逻辑简单而清晰——合并和分裂，这一点从代码上也容易看出。

实现：

我直接一把大锁（其实我真没想出能怎么拆）。

在具体的实现中，我采用调用 buddy_alloc 将维护 buddy system 的数据所在的内存分配出去，简化了初始化。

为了保证在内存不是恰好为 2 的整数幂时 buddy system 不会将有效内存之外的内存分配出去，同时不选取较小的 2 的整数幂作为 buddy system 维护区间的大小，导致无法利用整个内存区间，我先将 buddy system 维护的区间的末尾对齐到有效内存的末尾，并对齐到 16 MiB，然后将 buddy system 维护的区间的大小设置为能覆盖整个有效内存区间的 2 的整数幂，最后利用 buddy_alloc 总是将地址最小的内存先分配出去的性质，将开头的无效内存申请掉。

### 1.2. fast path: slab allocator

slab allocator 概述：

我采用栈来维护一块大内存上空闲的小块内存，优点是常数较小。

实现：

我给每个线程一把大锁。

我同时也准备了更加安全的方式，记录每块小内存的分配情况，这样可以在释放一块错误的内存时（给出的开头指针不对）给出警告。

### 1.3. 结果（小声）

用了两大法宝提速，当然想到底看看跑得快不快。

然后在我的电脑上测试了一下，typical workload， smp=4， 速度达到了 18Mop/s，通过 perf 发现此时速度瓶颈已经在测试框架上了，kalloc 和 kfree 总共只占了约 %4。

## 2. 基础设施

### 2.1. 测试框架

我搭建了一个测试框架，它可以根据不同的 work load 随机生成数据，并支持多线程和检查内存分配和释放的合法性以及测速。

### 2.2. minilib.h

为了提高代码复用，我将一些通用性较强的代码集中放在了这里。

### 2.3. rr + perf

这两个工具在我调试代码正确性和性能调优时帮了我很大的帮。

## 3. 印象深刻的 bug

### 3.1. 我的 slab 分配器，一直没上锁～

心情复杂～

其实 jyywiki 上写了的，但是由于 jyy 更新的太慢，我一直以为那些链接是点不开的。

### 3.2. 我 for 循环以 <= 为条件，然后上溢了～

心情复杂～

**正因如此，我意识到居然在初始化阶段死循环能过 3 个点！！！**（然后很奇怪为什么不是 0 个，也不是 6 个）

找不到的 bug 都在没有看到的角落里。

我一直关注的 buddy system 反而由于写的时候很认真，一个 bug 一没出。

## 4. 不足和反思

### 4.1. 测试框架

我的测试框架搭建地并不好，它无法有效地检测并发 bug。为了让线程可以释放别的线程取得的内存，我维护了一个数据结构存放所有已分配的内存，并通过一把大锁保护起来。由于我的这个数据结构写得极为平凡（很慢～），导致释放的测试都一个一个地从大锁里跑出来，降低了线程之间的并发性。

### 4.2. slab 分配器

我的 slab 分配器可以动态地从 buddy system 中获得空间（不足时继续申请），却不能动态地释放（大量空余时释放）。好在我们的测试中线程并不太多，即便大量空余，也不会造成太大的损失。