Lab2 - Allocator

实现思路

1. `kalloc_page`/`kfree_page`

   因为 page 的大小是固定的, 不需要额外保存大小, 所以可以用 free list 来保存所有空闲页.

   ```
   typedef ListNode Page;

   static Page dummy_page;
   static SpinLock page_lock;
   ```

2. `kalloc`/`kfree`

   使用 buddy 算法来管理块.

   块结构如下:

   ```
   typedef struct Block
   {
       usize order; // this field saved for both free and alloc block
       ListNode list;
   } Block;
   ```

   当被分配时, 需要保存块大小, 因此返回的是 `&Block->list` 的地址, 也就是说可以 `list` 在被分配时可以被覆盖.

   最小块的大小是 `1 << log2ceil(sizeof(struct Block))` (1^{5}B), 即 order-0 block 的大小. 最大块为 `PAGE_SIZE` (1^{12}B), 因此最高 order 为 8.

   ```
   static Block dummy_block[8];
   ```

   这种分配方式最差的内部碎片率为 $1/2$ (即全部分配 sizeof(order-N block) + 1), 平均为 $3/4$. 优点是分配的速度很快, 外部碎片少. *但是 benchmark 中只考虑物理页用量, 没有分配时间, 因此并不能取得很高的 Usage 分数.*

3. 并发

   在 `kalloc_page`/`kfree_page` 中使用 `SpinLock` 来保证并发安全.

   在 `kalloc`/`kfree` 中, 对比 `SpinLock` 和 cpu local 两种方法. (分别在 `git checkout f65e909eba7c42e6ae6f7742b0a16bdfeea0dd9a`, `git checkout 56ea57d8b953c36ed9158bbafb7d8169760a2d3b`)

使用 cpu local 的方法应该能取得更高的速度, 因为限定 buddy 算法中最高 order 块的数量后, 每个 cpu 独占的物理页会有上限, 因此 cpu 只拥有尽可能少的物理页. *但是这种优势在 benchmark 中没有体现.*