OSLab 实验报告

221840206_江思源

L1：物理内存管理(pmm)

代码架构设计

- 使用 buddy system 和 slab 共同实现物理内存管理，很好地实现了结合 workload 进行优化的目的。
  - buddy system 用于大块内存分配：
    大块内存在操作系统中分配的特点是：分配和回收的次数较少，同时一块内存多次使用。所以我们的需求是：不需要很快的分配速度，而需要方便的管理、回收和简单易实现的线程安全。
    buddy system：将内存按照2的幂次划分，每个块的大小为2的幂次，使用全局大锁。在获取（或释放）相应内存块时，只需要找到相应大小的块进行分割（或合并），对效率的影响不大，同时很好地保证了安全性。
  - slab 用于小块内存分配：
    小块内存在操作系统中分配的特点是：较为快速地分配和回收。所以我们的需求是：快速的分配（保证scalability）与回收。
    slab：根据内存大小划分不同 slab，slab 由 cache 管理，cache 持有 slab 链表以及 cache 的锁，slab 持有空闲块链表以及 slab 的锁。在获取（与释放）相应内存块时，只需要找到相应大小的 slab 进行空闲块分配（取空闲块链表头部分配并相应删除）或回收（链接至对应链表的头部），每个部分均有自己的锁，保证了线程安全、并发性和快速的分配。
  - slab 结合 buddy system：
    slab 中的 slab（大小设定为4096，为PAGE_SIZE） 由 buddy system 分配。

```c
struct free_list {
    struct list_head free_list;
    int nr_free;
};

typedef struct buddy_block {
    struct list_head node; // 用于空闲链表的节点
    size_t order;  // 2^order pages(页的阶数)
    int free;  // block是否空闲
    int slab;  // slab分配器，不为空时，表示页面为slab分配器的一部分
} buddy_block_t;

// buddy系统
typedef struct buddy_pool {
#define MIN_ORDER 0 // 2^0 * 4KiB = 4 KiB
#define MAX_ORDER 12 // 2^12 * 4KiB = 16 MiB
    struct free_list free_lists[MAX_ORDER + 1]; // 空闲链表（每个对应相应order）
    lock_t pool_lock[MAX_ORDER + 1]; // 保护伙伴系统的锁
    void *pool_meta_data; // 伙伴系统的元数据
    void *pool_start_addr; // 伙伴系统的起始地址
    void *pool_end_addr; // 伙伴系统的终止地址
} buddy_pool_t;
```

```c
// 空闲块链表
typedef struct object {
    struct object *next;
} object_t;

typedef struct slab {
    struct slab *next;        // 指向下一个slab
    object_t *free_objects; // 指向空闲对象链表
    size_t num_free; // 空闲对象数量
    lock_t lock; // 用于保护该slab的锁
    size_t size; // 每个对象的大小
} slab_t;

typedef struct cache {
    slab_t *slabs; // 指向slab链表
    size_t object_size; // 每个对象的大小
    lock_t cache_lock; // 用于保护该cache的锁
} cache_t;
```

精巧的实现

- 位运算技巧：
  - buddy system 中的合并获取相邻块的地址：通过位运算（addr ^ (1 << (block->order + PAGE_SHIFT))），获取相邻块的地址，再判断是否合并。
  - slab free：通过空闲块地址位运算（后12位置0）获取 slab 地址
- 通过 cache_t 来管理 slab：

    - 实现多级锁：cache_t 持有 cache 的锁，slab 持有 slab 的锁，在实现的时候思路更加清晰，并且保证了scalability。

印象深刻的bug

- 指针类型转换：有时候对一个地址进行不同的类型转换，再进行自加（减）运算会有不同的结果，一般会加（减）自己的类型大小。调试的时候让我对各种指针转换理解加深很多了。
- 指针没有对齐2^i：单纯的变量名写错了，导致 bug 找了好久，所以写代码使用更良好的命名风格很重要。