伙伴系统（Buddy System）物理内存管理器设计文档

1. 引言与概述

本文档旨在详细阐述一个为操作系统内核设计的伙伴系统（Buddy System）物理内存管理器。伙伴系统是一种经典、高效的内存分配策略，其核心目标是在处理频繁的内存分配和释放请求时，能够快速响应，并有效控制外部内存碎片。

该系统的基本思想是：将所有物理内存以 **2 的幂次方** 大小的块（Block）进行组织和管理。内存的分配通过将大块“分裂”成小块来满足，而内存的释放则通过尝试将相邻的“伙伴”块“合并”成大块来完成，从而尽可能地保持大块连续内存的可用性。

2. 核心数据结构

为了高效管理不同大小的空闲内存块，系统设计了以下核心数据结构：

2.1 free_buddy_t 结构体

该结构体用于管理某一特定“阶数”（Order）的所有空闲块。

```
typedef struct {
    list_entry_t free_list;   // 该阶数所有空闲块组成的双向链表
    unsigned int nr_free;     // 该阶数中空闲块的数量
} free_buddy_t;
```

2.2 free_buddy 全局数组

这是伙伴系统的中央管理枢纽，一个包含 MAX_ORDER (本项目中为 20) 个 free_buddy_t 元素的数组。

```
static free_buddy_t free_buddy[MAX_ORDER];
```

- **索引与大小的对应关系：**数组的索引 k 直接对应内存块的阶数。free_buddy[k] 负责管理所有大小为 2^k 个物理页的空闲块。
    - free_buddy[0] 管理所有 1 页 (2^0) 大小的块。
    - free_buddy[1] 管理所有 2 页 (2^1) 大小的块。

- free_buddy[k] 管理所有 2^k 页大小的块。

2.3 块信息的存储

一个空闲块的元数据（主要是其阶数）被巧妙地存储在该块的**第一个 struct Page 结构体**的 property 字段中。Page 结构体是物理页的“信息标签”，通过这种方式，我们无需额外开销即可获知任一空闲块的大小。

3. 关键算法与实现细节

伙伴系统的精髓在于其高效的分配（分裂）和释放（合并）算法。

3.1 初始化流程

buddy_init()

- **功能**：初始化 free_buddy 数组。
- **流程**：遍历整个数组，将每一个阶数的空闲链表初始化为空链表，并将空闲块计数器清零。

buddy_init_memmap(struct Page *base, size_t n)

- **功能**：接收一块任意大小的连续物理内存，并将其“分解”为符合伙伴系统规范的 2 的幂次方块。
- **算法**：采用**贪心递归**算法。
  1. **寻找最大块**：对于 n 页的内存，首先找出不大于 n 的最大 2 的幂次方 2^k。
  2. **创建主块**：将从 base 开始的 2^k 个页面作为一个阶数为 k 的大块，设置其首页的 property 为 k，并将其加入 free_buddy[k] 链表。
  3. **递归处理余料**：如果 n 不是 2 的整数次幂，则会存在 n - 2^k 的剩余内存。函数会**递归调用自身**来处理这部分剩余的内存。
- **示例**：初始化 7 页内存会被分解为：一个 4 页的块 (order=2)，一个 2 页的块 (order=1)，以及一个 1 页的块 (order=0)。

3.2 内存分配 - buddy_alloc_pages(size_t n)

此函数实现了“化整为零，按需分裂”的逻辑。

1. **计算阶数**：首先，计算出能够满足 n 页请求的最小阶数 k。

2. **查找空闲块**：从 free_buddy[k] 开始，向上（k+1, k+2, ...）查找第一个非空的空闲链表。
3. **分裂 (Splitting)**：如果是在一个更高的阶数 m (m > k) 上找到了一个可用块，则执行分裂操作： a. 将这个 m 阶块从 free_buddy[m] 中取出。 b. 将其**对半分裂**成两个 m-1 阶的“伙伴”块。 c. 将其中一个伙伴块（通常是地址较高的那个）**放回 free_buddy[m-1]** 的空闲链表中。 d. 对另一个伙伴块重复此分裂过程，直到其阶数降为 k。
4. **返回页面**：将最终得到的 k 阶块返回给调用者，并清除其首页的 Property 标志，表示其已被分配。

3.3 内存释放 - buddy_free_pages(struct Page *base, size_t n)

此函数实现了“化零为整，主动合并”的逻辑，是伙伴系统对抗外部碎片的核心。

1. **计算阶数**：计算被释放块的阶数 k。
2. **寻找伙伴 (Buddy Finding)**：这是伙伴系统最高效的部分。一个地址为 addr、阶数为 k 的块，其伙伴地址可以通过以下**位异或(XOR)运算**在 O(1) 时间内精确计算得出：
3. buddy_address = address ^ (1 << k)

   get_buddy_page 函数封装了此逻辑。

4. **循环合并 (Coalescing)**： a. 找到当前块的伙伴。 b. **检查伙伴状态**：确认伙伴块存在、空闲、且与当前块的阶数相同。 c. **如果满足合并条件**： i. 将伙伴块从其所在的空闲链表中移除。 ii. 将当前块与伙伴块合并成一个阶数为 k+1 的新块（新块的基地址是两者中较小的那个）。 iii. **以这个新合并的 k+1 阶大块为基础，重复步骤 a，继续尝试与它的新伙伴合并。** d. **如果不满足合并条件**：合并过程终止。
5. **加入空闲链表**：将最终形成的（可能经过多次合并的）块，添加到其对应阶数的 free_buddy 链表中。

4. 测试与验证 (buddy_check)

为了确保实现的正确性，buddy_check 函数提供了一套单元测试用例，覆盖了以下场景：

- **基本分配/释放**：验证小块内存的分配与回收。
- **地址与引用验证**：确保分配的页面地址唯一且初始引用计数为 0。
- **合并验证**：通过特定顺序的释放操作（如释放 1、2、4 页的块），验证它们能否正确地合并成一个更大的块（如 8 页）。
- **碎片整理验证**：模拟内存碎片化的场景，然后通过释放操作，验证系统能否将碎片化的内存重新合并为连续的大块。

5. 总结

本设计文档描述的伙伴系统物理内存管理器，通过基于“阶数”的空闲链表数组和高效的“分裂-合并”策略，实现了以下优点：

- **分配和释放速度快**：查找和操作的时间复杂度与阶数相关，性能很高。
- **有效抑制外部碎片**：主动的合并策略能持续地将小碎片整合成大块连续内存。

其主要缺点是存在**内部碎片**，因为所有内存请求都会被向上取整到最近的 2 的幂次方大小。尽管如此，它依然是操作系统内核中一种非常成熟和高效的物理内存管理方案。