Linux kernel-Buddy Allocator

==========

The Buddy Allocator is a memory management algorithm used in the Linux kernel to allocate and deallocate memory. It's a popular choice due to its efficiency, flexibility, and simplicity.

How it Works

The Buddy Allocator uses a tree-like data structure called a "buddy tree" to manage free memory blocks. The tree represents the available memory as a hierarchy of blocks with sizes that are powers of 2 (e.g., 16, 32, 64 bytes).

Here's a simplified overview of the algorithm:

1. **Initialization**: The allocator initializes the buddy tree by dividing the total system memory into blocks of a given size.
2. **Allocation**:
    - When an application requests memory, the allocator searches for the largest block that can satisfy the request.
    - If a suitable block is found, it's split into two smaller blocks: one to satisfy the request and another that remains free (its "buddy").
3. **Deallocation**:
    - When an application frees memory, the allocator combines adjacent free blocks with their buddies to create larger blocks.

Example

Suppose we have a system with 16 MB of memory. We want to allocate 4 KB of memory for an application.

1. The allocator searches the buddy tree and finds that there's a single block of size 16 KB available.
2. To satisfy the request, it splits the 16 KB block into two: one 8 KB block (which is allocated) and its "buddy" (also 8 KB), which remains free.

Example Code

```
 // Define the buddy allocator structure
struct buddy_allocator {
    void *start; // starting address of memory
    size_t size; // total system memory size
};

// Function to initialize the buddy allocator
void init_buddy_allocator(struct buddy_allocator *ba, void *start, size_t size) {
    ba->start = start;
    ba->size = size;
}

// Function to allocate memory using the buddy allocator
void *buddy_alloc(struct buddy_allocator *ba, size_t request_size) {
    // Search for the largest block that can satisfy the request
    void *block = find_block(ba, request_size);

    if (block == NULL) {
        return NULL; // out of memory
    }

    // Split the block into two: allocated and free buddy blocks
    split_block(ba, block, request_size);
}

// Function to deallocate memory using the buddy allocator
void buddy_free(struct buddy_allocator *ba, void *address) {
    // Combine adjacent free blocks with their buddies
    combine_blocks(ba, address);
}
```

This is a highly simplified example. In reality, the Linux kernel's Buddy Allocator uses more sophisticated data structures and algorithms to manage memory efficiently.

Notes

- The Buddy Allocator has an overhead of about 1-2% due to the tree structure.
- It's not suitable for systems with fragmented memory or frequent allocation/deallocation patterns.
- Other memory management algorithms, like slab allocators or buddy caches, are often used in conjunction with the Buddy Allocator to optimize performance.

Curated by Brajesh Kumar