Introduction

The segregated free list algorithm organizes free memory blocks into separate lists based on their sizes, aiming to efficiently allocate and deallocate memory.

Data Structures

We use a Block structure to represent a free memory block, consisting of its size and a pointer to the next block in the same list. The free lists are implemented as an array of pointers, and each list corresponds to a specific size range.

Macros

The ALIGNMENT macro is defined to be 8 to be inline with the Unix alignment (8-byte boundaries).

Initialization (void mem_init(size_t size))

During initialization, the system requests a large contiguous block of memory from the operating system and divides it into blocks for each free list. The starting address of this block is stored in allocated_block_start.

Memory Allocation (void* my_malloc(size_t size))

The my_malloc function determines the appropriate free list based on the requested size, searches for a suitable block in that list, and returns a pointer to it. If a suitable block is not found, the system may request additional memory from the operating system. If it is found, it allocates memory of size *size* bytes. It also uses alignment to ensure that memory is suitably aligned.

Memory Deallocation (my_free)

The my_free function marks a block as free and possibly merges it with neighboring free blocks to reduce fragmentation.

Testing

The mem_test.c file contains a simple test program that initializes the memory system, allocates and deallocates memory, and can be expanded for more comprehensive testing.

Performance

This implementation aims for efficiency by organizing free blocks into segregated lists, allowing for fast allocation and deallocation. The number of free lists and the size ranges they cover can be adjusted based on specific requirements.

Conclusion

The segregated free list algorithm provides a balance between simplicity and efficiency, addressing the general dynamic storage allocation problem. Adjustments can be made to further optimize the algorithm based on specific use cases and requirements.