Evaluation of Memory Allocation Strategies

Murathan Yelkovan Khaled Bhouri

February 9, 2025

1 Introduction

This report evaluates three memory allocation strategies: **First Fit, Best Fit, and Next Fit** implemented in a custom memory allocator. We compare their performance and memory usage with **Linux's default allocator (ptmalloc)**, which is highly optimized for general-purpose use. The primary focus is to analyze speed, fragmentation, and efficiency.

2 Custom Memory Allocator Implementation

The custom memory allocator redefines standard memory allocation functions (`malloc()`, `free()`, `calloc()`, `realloc()`) using a linked list to track memory blocks. The following allocation strategies are implemented:

- **First Fit**: Selects the first available free block large enough to satisfy the request.
- **Best Fit**: Chooses the smallest free block that fits the request to reduce fragmentation.
- **Next Fit**: Resumes the search from the last allocated block, potentially improving performance in some cases.

Memory fragmentation is controlled by merging adjacent free blocks upon deallocation.

3 Experimental Setup

To evaluate performance, we conducted three types of tests:

1. **Basic Tests:** Verify correct operation of memory allocation, reallocation, and freeing.
2. **Stress Tests:** Allocate and free a large number of memory blocks in a randomized manner.
3. **Performance Tests:** Measure execution time of 1,000,000 `malloc()` and `free()` operations.

4 Performance Results & Comparison

Allocator	Expected Speed
Linux ptmalloc (default)	Faster (∼0.05s)
Custom Allocator (First Fit)	Slower (∼0.88s)
Custom Allocator (Best Fit)	Slower (∼0.10s)
Custom Allocator (Next Fit)	Slower (∼0.96s)

Table 1: Comparison of allocation strategies

5 Why is Linux ptmalloc Faster?

The default Linux allocator (**ptmalloc**) includes several optimizations:

- **Thread-local caching:** Reduces contention in multi-threaded applications.
- **Segregated Free Lists (Binning Strategy):** Speeds up allocation by maintaining bins for different block sizes.
- **Lazy Freeing & Fast-bin Reuse:** Avoids immediate system calls by reusing freed memory quickly.
- **Minimized System Calls:** Avoids frequent `sbrk()` and `mmap()` calls.

6 Conclusion

The Linux allocator (**ptmalloc**) is optimized for general-purpose use, utilizing fast bins, caching, and binning strategies. The custom allocator, while functional, suffers from higher fragmentation and search times due to its linked list traversal. However, it can be optimized further for specific workloads.

Obviously, **Linux's ptmalloc remains the superior choice**.