ECE 650 – Spring 2025
Project #1: Malloc Library

Jingheng Huan
Duke University
Email: jh730@duke.edu

January 25, 2025

1 Introduction

Memory management is very important in system performance and reliability. The C standard library provides dynamic memory allocation functions like `malloc` and `free`, which are fundamental for managing memory in applications. This project1 involves implementing custom versions of these functions using two distinct allocation strategies: First Fit and Best Fit. The goal is to understand the underlying mechanisms of memory allocation and evaluate the performance implications of different allocation policies.

2 Implementation Description

This custom memory allocator is implemented in C, utilizing the `sbrk()` system call to manage the program's data segment. The allocator maintains a free list to track available memory blocks and employs metadata structures to store information about each block.

2.1 Data Structures

Metadata Structure: Each memory block is preceded by a metadata structure containing the size of the block, a flag indicating whether the block is free, and pointers to the next and previous free blocks.

Free List: A doubly-linked list that maintains all free memory blocks, which makes sure efficient traversal and management during allocation and deallocation.

2.2 Allocation Methods

2.2.1 First Fit

The First Fit strategy scans the free list from the beginning and selects the first block that is large enough to satisfy the allocation request. This approach tends to be faster as it requires fewer comparisons but may lead to higher fragmentation over time.

2.2.2 Best Fit

The Best Fit strategy examines all free blocks and selects the smallest block that is sufficient for the allocation request. This method aims to minimize wasted space within allocated blocks,

potentially reducing fragmentation but may incur higher computational overhead due to the exhaustive search.

2.3 Function Implementations

- `ff_malloc(size_t size)`: Implements the First Fit allocation strategy.
- `ff_free(void *ptr)`: Frees a memory block allocated by `ff_malloc` and coalesces adjacent free blocks.
- `bf_malloc(size_t size)`: Implements the Best Fit allocation strategy.
- `bf_free(void *ptr)`: Frees a memory block allocated by `bf_malloc` and coalesces adjacent free blocks.
- `get_data_segment_size()` and `get_data_segment_free_space_size()`: Provide metrics for data segment size and free space, including metadata.

2.4 Memory Management Strategy

1. **Allocation (`malloc`)**: Traverse the free list based on the chosen allocation policy (First Fit or Best Fit). If a suitable block is found, reuse it by possibly splitting it if it's significantly larger than the requested size. If no suitable block is available, extend the heap using `sbrk()` to allocate additional memory.
2. **Deallocation (`free`)**: Mark the block as free. Coalesce with adjacent free blocks to minimize fragmentation. Update the free list accordingly.

3 Performance Results

The performance of both allocation strategies was evaluated using three test programs that simulate various allocation and deallocation patterns:

1. **Equal Size Allocations**: Allocating memory blocks of uniform size.
2. **Small Range Random Allocations**: Allocating memory blocks with sizes randomly chosen from a small range.
3. **Large Range Random Allocations**: Allocating memory blocks with sizes randomly chosen from a large range.

The key metrics measured were execution time and fragmentation. Additionally, data segment metrics such as total size and free space were recorded.

Table 1: Performance Metrics for Equal Size Allocations

Allocation Strategy	Execution Time (s)	Fragmentation	Data Segment Size (bytes)	Free Space Size
First Fit (FF)	0.774383	0.456717	4,079,616	331,824
Best Fit (BF)	0.724615	0.456717	3,796,992	83,792

Table 2: Performance Metrics for Small Range Random Allocations

Allocation Strategy	Execution Time (s)	Fragmentation	Free Space Size (bytes)
First Fit (FF)	1.087728	0.081337	331,824
Best Fit (BF)	1.007308	0.022068	83,792

Table 3: Performance Metrics for Large Range Random Allocations

Allocation Strategy	Execution Time (s)	Fragmentation	Free Space Size (bytes)
First Fit (FF)	4.147240	0.114479	331,824
Best Fit (BF)	38.677297	0.040677	83,792

4 Analysis of Results

4.1 Execution Time

- **First Fit (FF)** generally exhibits fast execution times in the Equal Size Allocations and Small Range Random Allocations tests, likely due to the simplicity of scanning the free list until the first suitable block is found. However, in the Large Range Random Allocations test, FF's execution time increases a lot, indicating potential inefficiencies when handling a wide variety of block sizes.
- **Best Fit (BF)** shows slightly better or comparable execution times in the Equal Size Allocations and Small Range Random Allocations tests. However, in the Large Range Random Allocations test, BF's execution time increases substantially because of the higher computational overhead required to search for the optimal block.

4.2 Fragmentation

- **First Fit (FF)** tends to result in a relatively high fragmentation across all tests. This is expected as allocating the first available block can leave larger unusable spaces scattered throughout the memory.
- **Best Fit (BF)** achieves significantly lower fragmentation. This demonstrates BF's effectiveness in minimizing wasted space by selecting the most appropriately sized block for each allocation.

4.3 Data Segment Size and Free Space

- **First Fit (FF)** consumes more data segment space and maintains a larger free space size. This indicates that while FF may allocate memory quickly, it leaves behind more free space that is potentially fragmented.
- **Best Fit (BF)** utilizes the data segment more efficiently with a smaller free space size, reflecting its ability to consolidate memory usage more effectively.

4.4 Trade-offs

The analysis highlights a trade-off between execution time and memory fragmentation:

Table 4: Summary of Data Segment Metrics

Allocation Strategy	Total Data Segment Size (bytes)	Total Free Space Size (bytes)
First Fit (FF)	4,079,616	331,824
Best Fit (BF)	3,796,992	83,792

- **First Fit** offers faster allocation and deallocation operations in scenarios with uniform or small range allocations but at the cost of higher fragmentation, which can degrade performance over time as the free list becomes fragmented.
- **Best Fit** provides better memory utilization and lower fragmentation, which is beneficial for long-running applications or those with varied allocation sizes. However, it incurs higher computational costs due to the need to search the entire free list for the best-fitting block, leading to increased execution times in more complex allocation scenarios.
- For applications where allocation speed is critical and memory usage patterns are predictable or uniform, **First Fit** may be more appropriate.
- For applications where memory efficiency and minimizing fragmentation are paramount, especially with varied allocation sizes, **Best Fit** is preferable despite the increased computational overhead.

5 Conclusion

While First Fit offers faster allocation times in simpler allocation scenarios, it suffers from higher fragmentation, which can negatively impact performance in more complex or long-running applications. Conversely, Best Fit achieves better memory utilization and lower fragmentation, making it more suitable for applications with varied allocation patterns, with the trade-off of increased execution time due to its exhaustive search process. Understanding these trade-offs is essential for optimizing memory management in various application contexts.