Memalloc as an Alternative for Malloc

Fırat Kızılboğa
Computer Engineering
Istanbul Technical University
Istanbul, Turkey
kizilboga20@itu.edu.tr

***Abstract*—In contemporary computer systems, efficient memory management is pivotal for performance optimization. This paper introduces a bespoke memory management system that supplants the standard malloc and free operations with customized functions designed to better utilize system memory through various allocation strategies. The system initializes with a heap of a user-defined size, aligned to the system's page size using mmap. We then detail the implementation of MyMalloc, which supports four distinct memory allocation strategies: Bestfit, Worstfit, Firstfit, and Nextfit, enabling dynamic memory management according to user-selected criteria. Correspondingly, MyFree is used to deallocate memory and coalesce free memory spaces, thus optimizing subsequent allocations. The DumpFreeList function is utilized for debugging, providing a clear visualization of the heap's state at any moment. The system's robustness is demonstrated through controlled experiments where memory processes are dynamically allocated and deallocated, showcasing the effectiveness of each strategy and the overall system in managing memory efficiently. The results highlight the nuances between mmap and traditional malloc, with the former offering more precise control over memory allocation, essential for high-performance applications.**

I. Introduction

Efficient memory management is a cornerstone of high-performance computing systems, directly impacting application responsiveness and system stability. Traditional memory management techniques, while robust, often fall short in harnessing the full potential of modern hardware capabilities. This shortfall motivates the development of more adaptable and hardware-conscious strategies that can dynamically manage memory in ways that are both efficient and suited to specific application needs.

In this paper, we present a custom memory management system designed to replace the conventional malloc and free functions with MyMalloc and MyFree. These functions are tailored to offer flexible memory allocation and deallocation strategies that adapt to varied application demands and system states. The initialization of this system relies on mmap, a memory mapping technique that grants direct control over pages of memory, enabling precise and efficient allocation aligned with the system's page size.

The proposed memory management system introduces four allocation strategies: Bestfit (BF), Worstfit (WF), Firstfit (FF), and Nextfit (NF). Each strategy is designed to optimize memory usage in different scenarios, allowing users to choose the most appropriate method based on the current workload and memory state. This flexibility helps in minimizing memory wastage and improving allocation speed.

Additionally, the system features a debugging tool, DumpFreeList, which aids developers in visualizing the state of memory allocation at any point, providing insights into memory utilization and availability. This tool is crucial for ensuring the effectiveness of the allocation strategies and for fine-tuning system performance.

The development and evaluation of this memory management system are conducted in a controlled environment, using synthetic workloads to simulate various memory allocation and deallocation patterns. This approach not only demonstrates the capabilities of each strategy but also provides a comparative analysis that highlights the benefits and potential limitations of our system compared to traditional memory management techniques.

Through this paper, we aim to contribute to the ongoing discussions in the field of memory management and provide a practical solution that can be adapted for use in various high-demand computing environments.

II. Methodology

A. *Initialization: InitMyMalloc*

InitMyMalloc is the entry point for setting up the memory management system. This function first checks for valid input, ensuring the heap size requested is nonzero and aligns with the system's page size. It calculates the required heap size by rounding up to the nearest page size multiple using the getpagesize() system call.

Process:

Validation: Checks that the function is not called more than once and that the input size is greater than zero. Memory Mapping: Utilizes mmap to allocate the aligned heap space. Header Setup: Initializes a metadata header at the start of the heap to manage memory blocks. Outcome:

Returns 0 if initialization succeeds. Returns -1 on errors such as mmap failure or invalid parameters.

B. *Memory Allocation: MyMalloc*

MyMalloc is crafted to manage dynamic memory allocation through user-selectable strategies, enhancing flexibility based on application requirements. MyMalloc or the standard library function malloc are preferable over using the syscall mmap directly due to serveral reasons such as:

1) mmap allocatas a the systems page size directly, so it is quite inefficient when the programmer needs a lot of small pointers
2) mmap takes a lot more time due to its system call nature.

MyMalloc and malloc use mmap only when they need more than the current amount, then use clever accounting to make efficient allocations on both speed and space. Generally, mmap should only be called by malloc or, if the environment does not support malloc. MyMalloc looks for a free node using the given strategy when it finds a suitable location it inserts metadata (magic number for checking and size of pointer) and returns the pointer.

Strategies for efficient allocation:

Bestfit (BF): Allocates the smallest block that can fit the request to minimize waste. Worstfit (WF): Selects the largest block, potentially leaving a substantial free block after partitioning. Firstfit (FF): Quickly allocates the first block that is big enough, optimizing for speed. Nextfit (NF): Similar to Firstfit but starts searching from the end of the last allocated block. This function evaluates the heap, identifies the appropriate block based on the selected strategy, and modifies the heap metadata accordingly.

Return:

Provides a pointer to the allocated block or NULL if adequate space is not found.

C. *Memory Deallocation: MyFree*

MyFree is designed to free allocated blocks and consolidate contiguous free memory regions to prevent fragmentation and optimize subsequent allocations. MyFree's difference with standard library function free() is that MyFree will never use munmap syscall to give up heap, while there is a function in our library called "unmap()" for this functionality that uses munmap syscall, we chose not to use this at this time to have our allocator behave as an arena allocator.

Functionality:

Checks if the pointer is NULL and exits (checks magic number above the given location) if true to prevent errors. Updates the heap's metadata to mark memory blocks as free and merges adjacent free blocks. When a block is freed a node in the free list is created at that location and inserted in the order of starting location of nodes, not their actual location in memory.

Debugging Tool: DumpFreeList DumpFreeList serves as a diagnostic tool, providing a snapshot of the heap's current state. It lists all blocks, their sizes, and occupancy status, aiding in the visualization of memory usage and debugging.

This section succinctly yet thoroughly outlines the processes and functions of the memory management system, detailing how each component contributes to efficient memory handling.

III. Building the Project

To compile and set up the project for use on your system, follow the steps outlined below:

A. *Compiling with Makefile*

A `Makefile` is provided to facilitate easy building of the project. Open your terminal, navigate to the project directory, and use the following command to build the project:

```
make build
```

This command compiles the source files using predefined targets in the `Makefile`, ensuring that all necessary binaries are properly compiled and linked.

IV. Running the Project

After building the project, you can run individual programs with specified memory allocation strategies using the `make run` command.

A. *Executing a Program*

To execute a specific program with a designated memory allocation strategy, use the following syntax in your terminal:

```
make run CMD=P1 STRAT=0
```

This command runs the program 'P1.c' with strategy '0'. Under the hood, the 'make run' command performs the following operations: Here, 'clang' compiles the 'P1.c' file and links any required libraries, placing the output binary 'P1.o' in the './bin/' directory. The program is then executed with '0' as the argument, which represents the memory allocation strategy. Please build the project first before using this command.

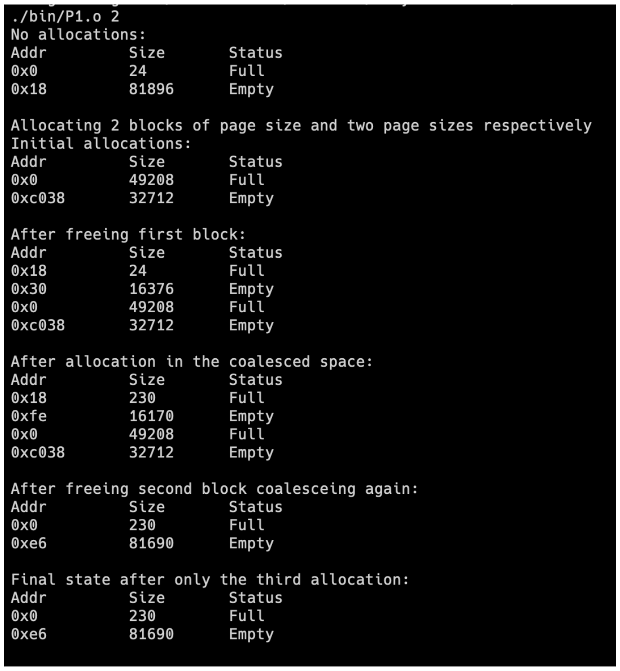
B. *Repository*

The entire project, including source files of the library and programs used for testing, Makefile is available on GitHub. You can clone or fork the repository from:

https://github.com/firatkizilboga/memalloc

V. Results

After building the project an running allprograms.c which spawns four different processes that uses MyMalloc here is a snippet of the output.



```
./bin/P1.o 2
No allocations:
Addr          Size        Status
0x0           24          Full
0x18          81896       Empty

Allocating 2 blocks of page size and two page sizes respectively
Initial allocations:
Addr          Size        Status
0x0           49208       Full
0xc038        32712       Empty

After freeing first block:
Addr          Size        Status
0x18          24          Full
0x30          16376       Empty
0x0           49208       Full
0xc038        32712       Empty

After allocation in the coalesced space:
Addr          Size        Status
0x18          230         Full
0xfe          16170       Empty
0x0           49208       Full
0xc038        32712       Empty

After freeing second block coalesceing again:
Addr          Size        Status
0x0           230         Full
0xe6          81690       Empty

Final state after only the third allocation:
Addr          Size        Status
0x0           230         Full
0xe6          81690       Empty
```

Fig. 1. Output for P1 using First Fit strategy

P1.c is very representative of the coalescing and reusing functions of MyMalloc as we can see both happening. See Appendix for the full output of allprograms.c.

VI. Conclusion

This report has detailed the development and testing of a custom memory management system designed to optimize allocation strategies like Bestfit, Worstfit, Firstfit, and Nextfit. While the system has demonstrated competence in managing dynamic memory allocation requests across various strategies, our analysis reveals that the coalescing mechanism requires further refinement to enhance its efficiency and effectiveness in real-world applications.

Additionally, the experimental unmap() function, currently in a beta stage, represents a promising enhancement to our memory management capabilities. Future work will focus on developing this function to fully integrate it into our system, aiming to improve the overall management of memory deallocation and system resource recovery.

Further research and development are necessary to address these areas, ensuring that our memory management system can meet the evolving needs of complex software environments more effectively.

Appendix

Complete Program Output

Listing 1. Complete Program Output

```
./bin/allprograms.o
Enter the memory allocation strategy
    ↪ number (0-3): 0

Executing P1.o with strategy 0
No allocations:
Addr            Size        Status
0x0             24          Full
0x18            81896       Empty

Allocating 2 blocks of page size and
    ↪ two page sizes respectively
Initial allocations:
Addr            Size        Status
0x0             49208       Full
0xc038          32712       Empty

After freeing first block:
Addr            Size        Status
0x18            24          Full
0x30            16376       Empty
0x0             49208       Full
0xc038          32712       Empty

After allocation in the coalesced
    ↪ space:
Addr            Size        Status
0x18            230         Full
0xfe            16170       Empty
0x0             49208       Full
0xc038          32712       Empty

After freeing second block coalesceing
    ↪  again:
Addr            Size        Status
0x0             230         Full
0xe6            81690       Empty

Final state after only the third
    ↪ allocation:
Addr            Size        Status
0x0             230         Full
0xe6            81690       Empty

Program P1.o (PID: 25904) completed
    ↪ with exit status 0

Executing P2.o with strategy 0
No allocationsAddr             Size
    ↪ Status
```

```
0x0           24         Full
0x18          16360      Empty

Allocating 4 blocks of size 3276 bytes
   ↪  using strategy 0
Addr          Size       Status
0x0           13192      Full
0x3388        3192       Empty

All allocations and deallocations
   ↪ successful.
Program P2.o (PID: 25905) completed
   ↪ with exit status 0

Executing P3.o with strategy 0
Addr          Size       Status
0x0           20584      Full
0x5068        28568      Empty

Addr          Size       Status
0x0           24         Full
0x18          49128      Empty

Addr          Size       Status
0x0           340        Full
0x154         48812      Empty

All allocations and deallocations
   ↪ successful.
Program P3.o (PID: 25906) completed
   ↪ with exit status 0

Executing P4.o with strategy 0
Initial allocations:
Addr          Size       Status
0x0           98392      Full
0x18058       81832      Empty

After freeing middle blocks:
Addr          Size       Status
0x4028        24         Full
0x4040        49160      Empty
0x0           98392      Full
0x18058       81832      Empty

After allocation in the coalesced
   ↪ space:
Addr          Size       Status
0x4028        24         Full
0x4040        49160      Empty
0x0           147560     Full
0x24068       32664      Empty

Final state after all deallocations:
Addr          Size       Status
0x0           24         Full
0x18          180200     Empty

Program P4.o (PID: 25907) completed
   ↪ with exit status 0
```