CSCI – 297 Systems Programming

Assignment 5

Due 11:59 PM, December 9, 2023

For this assignment, you will be implementing a simple memory allocator in C using an implicit free list as we discussed in class. You should write all your code in a file named **`my_malloc.c.`** You've been provided with the skeleton code **`my_malloc_skeleton.c`** which includes the functions you are required to complete. You should start by looking over the code, and make sure you understand the structure and flow of the functions.

In this implementation of **`malloc`**, we are using a simplified version of the heap, which in implemented using a static character array. As we know that a **`char`** is 1 byte, a character array of 1024, creates a heap of 1024 bytes. Note that we are not using any system calls such as **`sbrk()`** or **`mmap()`** to allocate memory on the real system heap.

The **`header_t`** struct represents a block header. This contains 2 fields. The **`size`** of the block and a Boolean value, **`free`** which indicates the allocated status of each block. The **`init_allocator`** function initializes the heap. It sets up the first block header at the beginning of the heap. This header block covers the entire heap (minus the size of the block header) is available and free.

The main functions to allocate and free memory, **`my_malloc`** and **`my_free`** have already been implemented for you. They use certain helper functions that you are required to implement. Details of these functions are listed below. You should implement a *find first* approach when implementing these functions.

- **`find_free_block(size_t size)`**
  - This function searches the heap for a free block of memory that is large enough to accommodate the requested size. If a block is found, the function returns a pointer to the block, otherwise it returns **`NULL`**.

- **`split_block(header_t *block, size_t size)`**
  - This function splits a block into two parts if it is larger than the requested size. The first part should be allocated, and the second part will be free. Note that the condition to split is already being checked in **`my_malloc`**.
  -
- **`coalese()`**
  - This function merges adjacent free blocks into a single larger block. This is done to reduce fragmentation and make better use of the heap space.

This is not a very long homework, and you can complete this easily with less than 50 lines of code. However, it is important to have a solid understanding of how implicit free lists work.

Hint: You will need to make use of pointer casting and pointer arithmetic to properly implement these functions, especially when integrating over the blocks in the heap. You should read up using online resources or recap our C/pointer lectures if you need to brush up on these specific concepts.

Submission Instructions:

You should only submit a single file: **`my_malloc.c`** in Canvas under the Assignment 5 tab.