G00870639 – Mrinmoy Mondal

G00870130 – Michael Pearlstein

CS367 Final Project Report

The Heap Memory Management project consists of the *mdriver*, memory header file, *mm\_free*, *mm\_malloc*, and *mm\_realloc*. All of these files were manipulated by the group in different degrees, with the main focus being *mm\_malloc*, *mm\_realloc*, and *mm\_free*. We will look at their structures beginning from the bottom with memory.h and go up in scope until we reach *mdriver*.

The *memory.h* file consists of the *mem\_rec* struct which contains the address of memory, size, upper and lower bound range, and next and previous pointers. It was seen fit to implement a single-linked interface for simplicity. A pointer seg\_free\_list in the mdriver class holds the Segragated lists. It is implemented using head pointers that then point each list if that list contains free nodes the head will point to the first one or it will point to NULL pointer. This allows for easy and simple deletion and assertions because the linked list, seg\_free\_list, contains a head per list. The functions defined within the header file, *mm\_maloc*, *mm\_realloc*, and *mm\_free* are what we will look at next.

The *mm\_malloc.c* file consists of three functions: *best\_fit*, *mm\_malloc*, and *split*. The *best\_fit* function, in layman’s terms, finds a block of space that can hold the size given. It finds the first empty block larger than it, which first requires going into the proper list that contains that range of size. Then it returns a head node that leads to this block for the memory to allocated. Next, we look at *split*. The purpose of split is to properly distribute the memory block given. So once *best\_fit* is called, the pointer returned is then sent into *split.* If the size required equals the number of blocks, it returns the node again. However, if the number of blocks within that list pointed to by the node is greater than the needed space, *split* will split the blocks, removing what is unnecessary for space and calls *mm\_free* on it. Once the excess is trimmed off, the head is once again returned. The last function within the file is *mm\_free*. The function aligns the proper size given, gets a best fit on size, then frees up what is unnecessary. Lastly, it returns the newly allocated space for use.

Now onto *mm\_free.*  The file contains the functions *mm\_free* and ­*coalescence*. *Coalescence* is first called when *mm­­\_free* is called. It finds every free space before and after the pointer given and pulls them together until it no longer can, creating a large span of empty blocks for proper use. Now as for the *mm\_free* function, it calls coalescence within a for-loop to defragment the empty space with all neighboring spaces. Then, inserts the empty blocks into the list in order of their addresses. Once accomplished, the upper and lower bounds are set accordingly to the segregated list it falls under.

The *mm\_realloc* file is quite like *mm\_free*. The file contains the *mm\_realloc* and *coalescence2* functions. *Coalescence2* functions quite similarly to the original, however it only checks addresses after the given head node. If there’s a chunk bigger than the space needed, it frees what’s left and allocates the rest. If the space is too small, it calls *mm­­\_malloc* to make up for the missing space. In the *mm\_realloc* function finds the single biggest free block after the given memory pointer’s address. Upon finding it, it will allocate it for what needs it. This works by cycling through the blocks after the head node. If the current node has less space than the current size needs, it moves off what isn’t needed to the segregated free list. If the new block is larger, it checks if the previous block can expand into it and get rid of the excess free space.

The last file is *mdriver.* Within it, *mm\_init* initializes all the segregated lists. It makes an array of pointers and initializes the lists off of them. After, they get the last segregated list, sets its address to zero, establish its upper and lower bounds, and puts all the blocks into the last list as they haven’t been allocated yet. The *print­­\_all* function just prints all the records. That is relatively self-explanatory. The *get­­\_set\_list\_info* function takes the information from the user about how the lists should contain their bounds and as for how many lists there are.

The program is very strong in the fact that whenever *mm­\_malloc, mm­\_realloc,* and *mm\_free* are called, they coalesce as many empty blocks as possible which leads to very organized memory that has little fragmentation. However, the downside is that since *coalescence* is run every time, the runtime complexity is much higher, translating into a slower program. We are currently not aware of any limitations or issues with the program, nor do we know of any extensions that we can apply to this.

Strength – whenever malloc, realloc and free are called, they coalesce as much as possible. Far less fragmentation.

Weakness – because it calls coalescence every time a malloc/realloc/free are called, it coalesces which is time consuming.