CS214 Systems Programming Assignment 1: ++Malloc

By: Rishi Shah and Eshan Wadhwa

Purpose:

The purpose of this assignment is to create an improved version of malloc and free using macros:

```
/* Redefining the malloc and free functions, using macros. */
#define malloc(size) mymalloc(size,__FILE__,__LINE__)
#define free(size) myfree(size,__FILE__,__LINE__)
```

The parameters that we included in our macro that differ from the actual malloc and free are __FILE__ and __LINE__. These are used to show the file and line where the error happened. These will notify the user of the program where the errors occurred with a cleaner interface than the original malloc and free.

Design Summary:

```
/* Linked List which contains the metadata of each block. */
typedef struct Node{
  short int size; // Size of the block.
  short int isFree; // 1=free, 0=in use
  struct Node* next; // Points to the next metadata block.
}Node;
```

The block of memory storing the metadata is stored next to the block of allocated memory which it refers to. The metadata block is required because we need to know if a block is free or not and we need to know the size of the block which it refers to. We decided to store the metadata in a Linked List. We have a short int size to indicate the size of the block, short int isFree to indicate whether or not the block is free, and a struct block*next to point to the next block. We have attempted the extra credit; the smallest size for the metadata we got was 12 bytes; 8 bytes for the next pointer, and 2 bytes each for the short int (isFree and size).

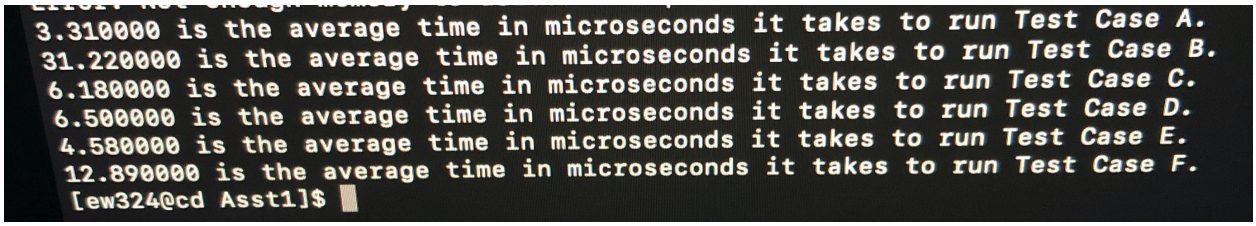
Explanation of mymalloc and myfree:

For mymalloc, we have 3 error cases. If the user requests to malloc less than 1 byte, then we return an error or if the user wants to malloc more bytes than allowed, we also output an error. We first initialize the memory, and then have a pointer that points to the beginning of the Linked List. Then we have a loop which keeps iterating through the Linked List until the pointer finds free space. If the metadata block checked refers to a chunk of memory that is the same size that the user requests, then we return a pointer to that chunk of memory. If the metadata block checked refers to a chunk of memory that is greater is of size greater than what is required, then we call the divide function which implements the First Fit Algorithm. This function starts from the first metadata block and searches for the first block that has enough space to allocate. We then return a pointer to that chunk of memory. After this process, we then have a third error case which is for the fact that there is not enough memory to malloc the pointer.

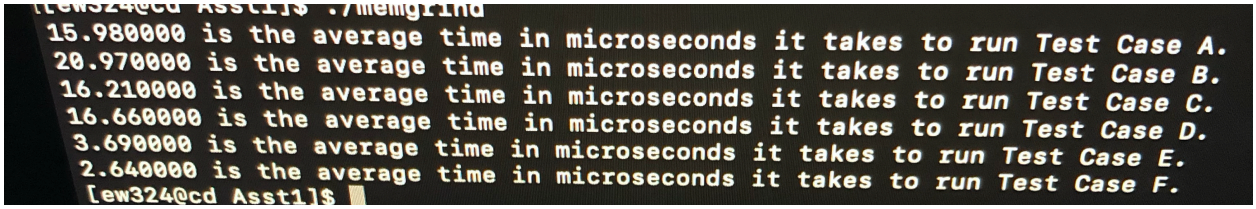
For myfree, we have 4 error cases. If the pointer that the user wants to free is NULL, then we output an error. We then check to see if the pointer lies within the heap so that it can be freed. If the pointer lies in the heap, we then have another error case to see if the pointer was malloced and to see if the pointer was freed multiple times. If it passes these cases, then we free the pointer. We call the combine function which combines the consecutive free blocks by removing the metadata in the middle. This helps to save space in our program. Without this function, mymalloc could return a NULL pointer, even though there is enough memory to allocate the pointer. After this process, we then have a fourth error which says that the pointer is not allocated by malloc.

Results:

The results of our program after running memgrind.c are:



Above are the results when calling mymalloc and myfree



Above are the results when just calling malloc and free

To find the results just for malloc and free we commented out the macros we made in the header file, and to find the results for mymalloc and myfree, we uncommented out the macros in the header file. On average, our implementation of malloc and free are more or less similar to the original malloc and free. There can be small discrepancies due to the fact with how we store our metadata and the fact that our mymalloc and myfree also has to run through error test cases which we coded, while the original malloc and free does not have to go through those test cases.

Summary:

This project made us create a more “smarter” malloc and free functions. In addition to the original functionality of malloc and free, now we get errors that say what file and line that error occurred in if there is an error when calling these functions. This is extremely useful for C because most of the time, when the user gets a segmentation fault, they do not know what went wrong and where it went wrong. To combat this problem, our mymalloc and myfree functions now output the error with what went wrong and it gives the file and line where the error occurred.