Cliff Chao and Ilana Zane

Metadata Implementation

- Metadata is of size 2 bytes. It contains a Boolean that determines whether the current space is already allocated, and the length of the pointer.
    - The Boolean is stored in the sign bit of the first byte (1 if already in use, 0 otherwise).
    - The pointer length can be any value from 0 to 4094 (block_size – metadata_size). The number gets split up and stored into the separate bytes.
    - The thousands and hundreds place digits get stored in the first byte, while the tens and ones place digits get stored in the second byte.
- For example, if the pointer length is 3469, then the number 34 (binary representation 00100010) gets logically ANDed with -128 (binary representation 10000000) to get -94 (binary representation 10100010). -94 is then stored in the first byte, while 69 gets stored in the second byte.
    - This bit masking does not cause conflicts because the pointer length cannot be negative.

MyMalloc Implementation

- The index being used always refers to the beginning of the allocated space (the pointer). The metadata is stored in the two bytes preceding the pointer's index.
- When malloc(x) is called, the function always starts by looking at location myblock[2].
    - It then looks for metadata in myblock[2 – 2] and myblock[2 – 1].
    - If the metadata is not in use (meaning myblock[index – 2] == 0 and myblock[index – 1] == 0), then the function will check to see if there is enough free space (defined as true only when the byte in question equals 0) from index to index + metadata.pointer_size.
    - If the condition is satisfied, then it will return the address of myblock[index]. Else repeat using index = index + metadata.pointer_size + 2 until either the condition is satisfied or the end of the array is reached.

MyFree Implementation

- The index is obtained by subtracting the address of the input pointer from the address of myblock[0]. Integrity of index is checked.
- The metadata is read from myblock[index – 2] and myblock[index – 1]. The integrity of the metadata is checked.
- If metadata.is_in_use is true, iterate through myblock from index to index + metadata.pointer_size, setting all data to 0. Then, metadata is set to 0, and the input pointer is set to null.

Workload Data

Case A average: 15.54 μs

Case B average: 83.21 μs

Case C average: 0.10 μs

Case D average: 0.20 μs

Case E average: 21.00 μs

Case F average: 39.59 μs

Workload Data Analysis

- Case B consistently produced the longest times elapsed, while Case C consistently produced the shortest times.
    - We believe this to primarily be because of the difference in number of malloc and free calls between the different cases allowing some cases to be done by faster caches than others.
        - For example, each iteration of Cases A and B required 150 malloc and free calls each, while Cases C and D only required 50 malloc and free calls (free(void*) was not called if there was no existing malloc'd pointer).
- The cases where the pointer was freed immediately (Case A, C, and E) produced faster times despite allocating the same amount of memory as the cases where the pointers were all freed at the very end (Case B, D, and F).
    - We believe this to be due to increased temporal locality in Cases A, C, and E giving them better times despite similar levels of spatial locality.
        - This difference seems to get further accentuated the larger the amount of function calls gets. The time difference between Case A and B (150 calls each) is much larger than it is between Case E and F (50 calls each).