Umalloc!

Names: Vennela Chava(vc494), Sai Laxman Jagarlamudi(sj1018)

Objective

To build a library that will serve malloc() and free() calls using a static character array that is 10MB long.

Design

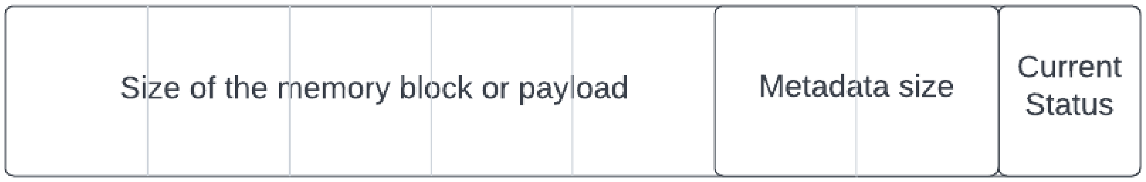
We reduced metadata space as much as possible in order to optimize actual available memory space. To achieve minimal metadata space, we employed dynamic size for metadata. The size of the metadata is determined by the amount of memory space requested by the user. In our project, within malloc() and free() we also addressed improper use of malloc() or free() and returned their respective diagnostic error messages.

Metadata

The data that provides information about other data is Metadata. In our project metadata has the information about size of the metadata of a memory block, current status of the memory block(i.e., if the memory block is used or not), size of payload or memory block. As mentioned above to contain the above information within the metadata, a metadata has

- One bit(least significant bit) has the status of the memory block
- Two bits are used for the size of metadata. Metadata in our project can be of three sizes depending upon the requested size of the memory block. The sizes are 1B(represented using 00), 2B(represented using 01) and 4B(represented using 11). We are considering three metadata structures depending upon the memory block, payload size requested by user.
- The remaining all bits consists of size of the memory block or payload.
    - If the size of the metadata is 1B then 5 bits(1B=8bits->8bits-3bits) are used to store the size of the memory block+metadata. The maximum possible size of the memory block+metadata(1B) in this case is 31B(i.e., 2^5-1).
    - If the size of the metadata is 2B then 13 bits(2B=16bits->16bits-3bits) are used to store the size of the memory block+metadata. The maximum possible size of the memory block+metadata(2B) in this case is 8191B(i.e., 2^13-1).

- If the size of the metadata is 4B then 29 bits(4B=32bits->32bits-3bits) are used to store the size of the memory block+metadata. The maximum possible size of the memory block+metadata(4B) in this case is 536870911B(i.e., 2^29-1).



Example

Assume if the user requested for memory block of size 24B. Then the current status of the memory block or payload will be set to 1. Since the memory size requested by user is less than 31B, the size of the metadata will be 1B and it will represented as 00 in the metadata. The remaining bits are used to represent the size of the payload or memory block here which is 24B represented as 11000. Thus our metadata will be [11000|00|1].

Malloc

Our algorithm for the malloc uses segmentation and first-fit selection strategies. When a user requests for a block of memory we linearly search the memory for the contiguous block that can fit the requested memory. When the block is found the malloc returns the pointer to the allocated memory block. If such a block is not found we return the memory error depending on the situation.

- **Case 1:** There is no free memory. That is the memory is completely allocated and memory of any size can not be allocated further.
- **Case 2:** Not enough memory for the allocation.
- **Case 3:** Though there is enough free memory there is no contiguous block available for the allocation.
- **Case 4:** If the requested memory size is not valid i.e 0 or less.

Free

We don't have much flexibility in terms of releasing memory because of the restricted metadata space we're working with; nonetheless, our technique still allows us to free appropriately and catch any potential errors. Because of our dynamic metadata size, we cannot instantly calculate the size of the allocated memory given the pointer to the memory address. To address this issue, we linearly search through the whole memory, keeping a reference to the beginning of the metadata and determining where the memory is. If the pointer to where the memory would be matches the pointer supplied to us as parameter to free(), we can confidently assert that this is the memory the user wishes to be freed. Otherwise we would land in one of the following cases:

- **Case 1:** If you attempt to deallocate a NULL pointer then free() throws error.
- **Case 2:** If you are trying to deallocate already deallocated memory address then free() throws redundant error.
- **Case 3:** If you are attempting to deallocate using an invalid pointer then free() throws deallocation error.

Utility Functions

For implementing malloc() and free() functions we defined the following helper functions:

- **currently_using:** It takes pointer to metadata as input parameter. It is used to track the usauge of current memory block. Returns 1 if the memory block is inuse else returns 0.
- **metadata_size:** Metadata pointer is taken as the input parameter. It is used to calculate the size of the metadata based on the memory size requested by the user.
- **payload_chunk:** The pointer to metadata is taken as input. It is used to calculate the size of the payload or memory block using the metadata pointer.
- **payload_size:** The total size(i.e., metadata+memory block size) is taken as the imput parameter. It calculates and returns the size of the memory block or payload.
- **set_metadata:** Metadata pointer is taken as the input parameter. It populates the metadata with the appropriate information.
- **unset_metadata:** Metadata pointer is taken as the input parameter. When called it makes the least significant bit of the metadata(i.e., status of the current memory block or payload) to zero.

Performance

Various functions are defined to check the performance of malloc() and free() functions. The function descriptions and results are as follows:

- **Consistency:** Used to check if the addresses of two pointers are same. Ptr1 is generated by allocating small block and casted it to type int after which wrote integer 3 into location pointed by Ptr1. Freed the previous pointer(i.e Ptr1) and repeated the same process on Ptr2. Found that Ptr1 is consistent with Ptr2(i.e., Ptr1=Ptr2). Proving the consistency of malloc() and free().

- **Maximization:** In this, initially 1B of allocation is done using malloc(). The current allocation is freed using free() and double the previous allocation. This is repeated until malloc() returns NULL then half the size is taken and deallocate it and repeat the process again. With this process it is found that maximum memory that can be allocated using malloc() is 8388608.
- **Basic Coalescence:** Quarter of maximum memory(i.e., 8388608/4) is allocated followed by half of maximum memory allocation(i.e., 8388608/2). Then the current total allocation is freed. After this, we were successful in allocating maximum memory.
- **Saturation:** Using loop 9K 1KB allocations are made. Switched to 1B allocations until malloc() returns NULL. It is found that after 515072 allocations of 1B the saturation space reached. The average time taken for reaching saturation space is 1100sec~ 18.33mins.
- **Time Overhead:** After saturation, the last 1B of memory is freed and reallocated. The time taken for the reallocation is called the maximum time overhead. In our case on average, maximum time overhead is 4314 micro secs or 0.0043 secs.
- **Intermediate Coalescence:** During saturation all the pointers returned by malloc() are stored in an array. This array is used in the intermediate coalescence function to free the memory space as an intial step. After which, our attempt to allocate the maximum memory is successful.