Memory Allocator

1. Design of Allocator

2.1 Common Architecture

- In the my_setup() function, we initialized a global configuration struct, in which we stored malloc type, memory size, starting address of the memory, header size, maximum and minimum size of the chunk, and objects requested per slab. Also, we pre-calculated a sizeArray which provides the 2^index. Additionally, we created a buddy block of the maximum size provided.
- When my_malloc() function is called, the header size is added to the requested size by the user. The pointer, which is returned, returnLocPtr, is set to the address of the memory again after the header size offset of the beginning of the chunk allocated. This results in the true address and size of the block at the input determining my_free by looking at the header of the pointer.
- The allocators have different structures, like BuddyBlock, which stores information of a single buddy block, BuddyConfig stores the bucket heads, SingleSlabConfig stores the metadata of a single slab, SlabTypeConfig maintains the metadata of a type of slab, and lastly SlabAllocator maintains the linked list of all slab types.
- my_cleanup() function iterates through all the linked lists, like slab-type lists, allocated block list, bucket heads list, made inside the project, and frees all their nodes and bitmaps inside the slab.
- **NOTE: Whenever we are talking about power, assume it is the power of 2.**

2.2 Buddy Allocator

- We designed a solution in which we have multiple buckets based on the power of 2(size of blocks). We try to move the buddy blocks in between these buckets. These bucketheads are created as a linked list of BuddyBlock nodes, which are free. The lists are kept in increasing addresses to imply the lowest starting address selection rule and effective merging. Another list, allocatedLLHead, is used to record all currently allocated blocks to be able to look them up quickly when my_free() is called.
- **Allocation (buddyAllocation):** In this function, we first calculated the total size of the block, then calculated the required size for the same. Further, we did a lookup for the bucket, based on power, which has empty blocks. We find the smallest available block(smallestBucketPower). If a bigger block is discovered, it is split (halved) till we don't reach a size just greater than or equal to the desired size. The leftBuddy (lower address) is allocated, and the rightBuddy in the next smaller size bucket is put into the free list. The BuddyBlock is set as in use and inserted in allocatedLLHead at the start. Lastly, the address after the header for the block just added inside the allocated list is saved to be used for freeing purposes and is finally returned.

- **Deallocation (buddyFree):** The BuddyBlock to be freed is found through ptr in the allocatedLLHead and is deleted from that list. This block is marked as free. The code tries to merge this block with its free potential buddy recursively. The address of the potential buddy, buddyStart is calculated based on the size of the current block and the start address of that block buddyStart is the start address of the neighboring block that would create a block of the larger size of twice the current block. We try to figure out if the potential buddy is a left or right buddy. When the potential buddy is found to be free in its appropriate bucket, the two blocks are combined into one block twice the size, which is achieved by removing the potential buddy and extending the current block. This merging is repeated at the next level (power) bucket.

2.3 Slab Allocator

- It is constructed above the Buddy Allocator, as per the project requirements.
- We have coded in such a way that it holds slabTypes, which is a linked list of SlabTypeConfig nodes, each of it representing a particular requested object size. Basically, we are creating slab types based on the object sizes inside them.
- SlabTypeConfig contains the size of each slab, and two linked lists of nodes (SingleSlabConfig) as emptySlabList (partial slabs), and the fullSlabList (full slabs).
- SingleSlabConfig implies a slab of memory, and this has also been allocated through the Buddy Allocator. It holds the starting address, number of free slab objects, and a bitmap to indicate the allocation status of objects inside the slab.
- **Allocation (slabAllocation):** It searches for the type of slab for the size inside the existing SlabTypeConfig. In case it doesn't exist, a new one is formed. It tries to assign the slab to an emptySlabList. In case there is no empty/partially empty slab list, the createNewSlabForType() creates a new slab.
    - createNewSlabForType() calculates the total slab size for *n_objs_per_slab* objects and allocates the memory using the **buddyAllocation()** call. Each object is assigned a slot (bit 0) using a bitmap created using calloc. This new slab is added to the empty slab list.

  We further look for an empty spot inside the slab using a bitmap to allocate memory with the lowest memory address. When the slab has been filled, the slab is removed from the emptySlabList into the fullSlabList. The returnLocPtr is computed with the help of the address of the slab start, the size of the object, and the index.
- **Deallocation (slabFree):** In this function, we only have ptr; therefore, we need to find the slab-type, the slab, and the object index. locateSlabUsingPtr() is used to find the containing slab and the index of the object in the slab. The bit in the bitmap of the object is cleared, and the number of freeSlabObjects is increased. In case the slab has already been full (and now has 1 free object), the slab is moved from fullSlabList to emptySlabList. Else if the slab becomes completely free, we remove the slab from emptySlabList, and this slab acts

as a buddy, which is just removed from the allocated list and is ready for searching for its buddy and merging.

4. Key Implementation Choices

- The pushBlockInBucket operations will make sure that free blocks will be inserted sequentially by increasing the start address as directed in the project requirements, which are to always select the lowest starting address to allocate free blocks.
- The value of MAX_POWER according to project constraints should be around 25, but for safety purposes, we kept it as 30.
- The locatePtrIn and buddyAllocation functions properly apply to the use of pointer arithmetic by casting void or unsigned char is the use of the cast to calculate the memory offsets (byteDifference) and returning addresses (rel). As said in the recitation PowerPoint, a void doesn't have a size, so we cast the pointers to uint8_t, which is basically an unsigned char having a size of 1 byte.
- The control structures we used in our code, such as the BuddyBlock, SlabTypeConfig, and others, are directly malloced and freed with malloc () and free (), respectively, based on the standard C library. We were required to use calloc for initializing the bitmap because calloc, by default, initializes all values to 0.
- We initially thought of all the creation of buckets and handling of blocks/slabs inside them using sizes. But the calculation became very tricky. So, we switched to powers.

5. Optimizations

- We initially wrote a function to calculate power from size, size from power, and requiredSizeFor(). But we found out that over the course of a single input, we are required to calculate the powers multiple times. To reduce the calculations, we created a sizeArray with indexes till MAX_POWER that pre-calculates the power of 2. Whenever we are required either a size of block/slab or power for the bucket, we simply need to use sizeArray instead of repeated calculations.
- Initially, we created a single list of slabs in a slabType, and it worked too. But when we were trying to look for an empty spot to allocate space to an object inside a slab, we were required to traverse all the slabs. So, we separated the full slabs. Hence, we need to iterate only through the bitmap of empty slabs.

6. Known Limitations / Not Implemented

As much as we know, we are not aware of any limitations. We tried to cover most of the edge cases.

7. Testing Summary

- We tried running all the inputs given in the GitHub repo for both Buddy and Slab allocator. All the cases are passing, and we also tried running multiple times for all inputs. The outputs are consistent on Lab 135 machines.
- Also, we tried to run inputs from the project details, and we were successful in running all the edge cases.