Memory allocator

January 11, 2020

The goal of this assignment is to implement a memory allocator (`SimpleMM`) for C applications. `SimpleMM` provides interfaces for dynamic memory allocation and deallocation.

1 OS memory interfaces

`SimpleMM` uses `alloc_from_ram` and `free_ram` APIs for allocation and deallocation of RAM space from OS. `alloc_from_ram` can be used to allocate memory of size multiple of 4096 bytes. The memory address returned by `alloc_from_ram` is always aligned to 4096 (i.e., the address is divisible by 4096). A memory area of size 4096, whose starting address is aligned to 4096, is called a page. In other words, `alloc_from_ram` can be used to allocate contiguous page(s). `free_ram` API can be used to deallocate memory returned by `alloc_from_ram`.

2 Allocation

`mymalloc` is the memory allocation API of `SimpleMM`, which is similar to `malloc` API of standard C library. `mymalloc` takes the size of the buffer as input and returns a memory buffer of the input size. `SimpleMM` maintains multiple lists. Each list contains free objects that are available for allocation. All objects in a given list are of the same size. A list of free objects is also called a bucket. `SimpleMM` maintains nine buckets that contain objects of size 16, 32, 64, 128, 256, 512, 1024, 2048, and 4080 bytes, respectively. Bucket size is the size of memory objects in a given bucket.

If the allocation size is less than or equal to 4080 bytes, then the allocation size is rounded up to the nearest bucket size, and the allocation takes place from the corresponding bucket.

If the bucket is empty, then `mymalloc` allocates a page (using `alloc_from_ram API`). The first 16 bytes on the allocated page is reserved for metadata; the rest of the page is called data area. All objects on a page are used by the same bucket. Page metadata contains the bucket size and the number of available bytes on a given page (i.e., the number of bytes on the page that the available for allocation). `mymalloc` divides the data area on the page into memory objects of bucket size and inserts them to the bucket. After this step, `mymalloc` removes an object from the bucket, updates the available size in the page metadata, and returns the object to the caller.

If the bucket is not empty, then `mymalloc` removes an object, updates page metadata corresponding to the page of the object, and returns the object (similar to the last step in the previous case).

If the allocation size is larger than 4080, then `mymalloc` always uses `alloc_from_ram` to serve the memory request. `mymalloc` also keeps page metadata for large allocations (needed during deallocation). The input allocation size is adjusted to also accommodate page metadata at the beginning of the page returned by `alloc_from_ram` (similar to small allocations). Finally, `mymalloc` returns the memory address just after the page metadata after updating the page metadata.

3 Deallocation

`SimpleMM` memory deallocation API is `myfree`. `myfree` takes a memory object (allocated using `mymalloc`) as input. `myfree` fetches the page metadata, which is stored on the top of the current page (page corresponding to the input object). If the current page belongs to the large allocation (>4080 bytes), then `myfree` obtains the allocation size from the page metadata and immediately frees the page using `free_ram` API. Otherwise, `myfree` updates the available size on the page and inserts the object to the corresponding bucket (list). If all the bytes on the current page are available, then `myfree` removes all the objects on the current page from the corresponding bucket and frees the page using `free_ram` API.

4 Implementation

You are to implement the `mymalloc` and `myfree` APIs, as discussed in Section 2 and Section 3. You are not supposed to use the standard memory allocator `malloc` and `free` anywhere in your implementation. You have to implement your own linked list yourself. The third-party linked list libraries are not allowed. You can use `alloc_from_ram` and `free_ram` APIs, which are provided in the assignment repository. You need to implement everything in `‘‘memory.c’’`.

5 Environment

For this assignment, you need to clone the assignment repo from `https://github.com/Systems-IIITD/SimpleMM`.
`SimpleMM` contains a test case `randomalloc.c` and the memory allocator(`memory.c`). You are to implement `mymalloc` and `myfree` APIs in `memory.c`. `memory.c` contains implementations of `alloc_from_ram` and `free_ram` APIs. To run the

test case, run ``‘‘make run’’``. “`make`” command builds the test case and the `SimpleMM` library. You are not supposed to change the test case.

5.1 Design documentation

You also have to submit design documentation along with your implementation; otherwise, the assignment will not be graded. Answer the following questions in your design documentation.

- What is your page metadata structure?
- How do you find that an object is large or small during `myfree`?
- When do you free a page allocated for objects in buckets (lists)?
- How do you find the page metadata of the input object during `myfree`?
- Paste your code corresponding to the removal of all objects on the page from the bucket (list), when a page is freed.
- Dump the output of the ``‘‘make test’’``.

5.2 How to submit.

To be done individually. Submit a zip folder that contains two files: “memory.c” and design documentation (in pdf format). The submission link is on backpack.