Bitnine Memory Manager (BMM)

Memory Management is part of complex softwares. Memory Manager is given a fixed size memory block called memory pool. Different modules in that software request different-sized blocks of memory from Manager. Manager uses first-fit algorithm to get requested blocks from memory pool. Upon finding the first-fit block from memory pool, Manager passes it to the requesting module. When a module does not need the allocated memory anymore, the memory block is returned to Manager so that Manager merges it back into the memory pool. This is done so that Manager can allocate it to some other module on request.

Write C language (not C++) to implement your Memory Manager. Your code should be well written. You will be judged for code quality, code comments, and code readability. Your software design will be judged for modularity of design, design technique and how optimzed it is.

Functional Requirements

1. Manager should provide a well-defined application programming interface (API) for initialization. (The total memory to be used by Manager should be provided to this API)
2. Manager should provide an API to allocate a chuck of memory, same like the traditional malloc().
3. Similarly like the free(), Manager should provide an API to de-allocate memory.
4. All provided APIs should return SUCCESS/FAILURE status. For example, if the memory allocation API is not able to allocate anymore memory due to lack of memory in memory pool, it should return FAILURE
5. Design and implement a testing system (test-bed) for your Manager.
6. The test-bed should initialize Manager memory pool by calling the initialization API and providing it memory pool size for example 512 MB.
7. The test-bed should be able to randomly allocate and de-allocate fixed and variable-sized memory blocks using API calls provided by Manager. Test ending criteria is left for you to decide.
8. After every complete run, the test-bed should provide the following statistics:
    - Manager memory pool size for the test run.
    - Total number of memory allocations and de-allocations.
    - Average memory allocation and de-allocation time.
    - Maximum, minimum and average size of memory chunk allocated in the test.
    - Total number of failed memory allocation request in the test.

Deliverables:

1. Application source code.
2. ReadMe file with compilation instructions, how to run the program, how to test the program.
3. Output for the following test runs:
    - Pool Size: 2 MB Block size per memory allocation request: 5 KB
    - Pool Size: 1024 MB Block size per memory allocation request: 2 MB
    - Pool Size: 1024 MB Block size per memory allocation request: Variable Sized
4. Brief write up on your design and the reasoning behind your choices of the algorithms/data structures. If you want to include diagrams, you are allowed to include scans of hand drawn diagrams to save time (keep scanned file sizes small).

Suggestions:

- Break up the problem and tackle it one step at a time. At the completion of each step, you will have a working system which will fulfill part of the requirements.
- If you cannot complete the entire solution, submit the parts that you have completed provided that they are in a running condition.
- This is an opportunity to display your design and coding skills, please do the best that you can.