Operating Systems and Kernel Design

EECE 4811/5811

Prof. Megherbi

Fall 2021

Assignment 3

In this assignment, you will implement a Memory Management System (MMS).

Your MMS will handle all requests for memory space allocation by different users (one thread per user).

Your MMS will provide the user with an interface for making memory requests and also for freeing up memory that the user no longer needs. One of the tasks of your memory management system is to service user memory requests by matching the size of the request with a large enough space from which to satisfy the request. Make sure that the size of the memory allocated is in bytes and is in a power of 2.

When your MMS gets the request of size *"size",* it will allocate a memory space with a size "size" memory chunk requested. Write the code and use two functions, one called *memory_malloc* (which calls the system function malloc that you are familiar with) to allocate memory and the other *memory_free* to de-allocate the memory. Make sure to know the size of what is being freed. Every time a block of memory is allocated by *memory_malloc,* store the size of that block (as an integer) somewhere. Note that *memory_malloc* returns a pointer to the first byte of the block of memory it just allocated. The size of the block allocated will be used when the user calls *memory_free* to return the block to the free memory blocks.

As further explained below, you need to implement your code in two parts, **part-1** of the case without fragmentation and **part-2** of the case with fragmentation to merge freed memory blocks (fragmentation) together to give one single block hole of a larger size in the MMS thread function before allocating any memory block to the users.

First, use a standard malloc call to allocate a large amount of memory from the system. The specific amount of memory that your system will be able to handle will be defined by the parameter *MAX_ZISE*. This allocated memory will form free memory blocks, and allocated blocks of memory to the user are taken from here when your function that you will write is called *memory_*malloc that will call malloc with a specified requested size from a user thread.

Your *memory_malloc* should call one of the three functions *First-fit*, or *Best-fit* or *Worst-fit.* To implement these functions, please follow the definition of each one of them, as seen and explained in class. To simplify things, you can write three separate similar programs, one for each of *First-fit*, *Best-fit* or *Worst-fit*.

Your program should have one command-line argument which specifies the number N of threads (users). When the program starts, it creates N+1 threads which place themselves in a thread pool. N of these threads will be the users that request memory space from your MMS. The MMS is implemented as a separate thread, that is, the remaining of the N+1 threads. Make sure to pay attention to thread synchronization issues, if any.

Work in a multithreading concept where each thread will request the MMS for some memory space, print on the screen "*I am thread #thread-ID going to sleep*", then sleep for some random time and then wakes up and prints on the screen "*I am thread #ID, waking-up*", and then ask the MMS to release that memory block previously allocated to the thread. Make sure you cover the case where the MMS runs out of memory space. In this case, it will choose one or more thread users, treat them as the ones with the lowest priority, and force them to give up their allocated memory. Make sure that you cover the case where you need to merge freed memory holes together (fragmentation) to cover a larger size needed by some user (s), as described in paragraph-6 above. Make sure to cover the case where a very large size requested cannot be accommodated by the system. In this case, the requesting thread shall be notified with an error notice and the need to request a size no larger than a specified Max size that you specify/consider when designing your MMS and return said thread (s) to the pool of threads, to compete/try again among the other threads in the pool.

Note: Your readme file shall explain how to run your code with every of the specified three functions, *First-fit*, or *Best-fit* or *Worst-fit,* as well in the case of fragmentation and in the case without fragmentation.