Oytun Kuday Duran 28357

First of all, I compiled my program with the following code:

g++ -g -o main <samplerunnumber.cpp> allocator.cpp -lpthread; ./main

PA 4 Report:

Locking Mechanism and Concurrency:

In the homework, Firstly, I thought about using one lock per node to increase concurrency. Then, while mergeing multiple blocks or allocating, since it had a chance to deadlock, I instead used a global pthread_mutex_t lock and declared it as "PTHREAD_MUTEX_INITIALIZER". When going in a function that threads use, I immediately lock the global lock so that there won't be any race in the linked list or while checking the values, they won't read wrong values and won't make a contagious list. I didn't use the lock in print() function since, it is already called inside other functions while in a locked state. I also didn't use it in constructor,initHeap and destructor since they are also not called by threads. So far, I created text cases with around 10-20 threads and memory value of 200-300, and there were no problem about data race or atomicity since only one thread at a time can use a function that uses our linked list. It satisfies atomicity since all the function that a thread calls goes without any interruption of other processes that change the values or edit linked-list. So, the functions that are called for threads are isolated from other operations that are occuring at the same time because of other threads. They have to wait the end of a process of thread to edit the data structure themselves by locking the global lock. Unless a thread achieves the global lock, it can't do anything operations on the linked-list structure. In other words, the thread owner is isolated well.

My program Implementation and Hierarchy:

In my class named HeapManager inside allocator.cpp, I used my own linked list implementation. I created a struct named HeapNode which the linked list consists of it's pointers. It has all the datas that are mentioned in the PA4.pdf file such as size,id or index of the memory block. It also has a pointer that points to the next block which we are obliged to have to create a linked-list. I also created 2 constructors to make new operations easier.

In the class, I have 2 private members. The first one is head and initialized as the NULL. We also keep this value to reach our linked-list. It is the start of the linked-list and we Access other blocks inside linked-list by iterating through head. Secondly, we also have the global pthread_mutex lock that we use to ensure atomicity, isolation and safe-concurrency.

In public part, I have constructor and destructor. The constructor allocates memory for the head node with proper values. In destructor, we simply delete the allocated memory for linked-list to reuse again later when we don't need the linked list structure later. They are also not called by threads so I didn't use any locking mechanisms. Similarly in initHeap, I didn't lock since it is executed only once and without the threads, we simply change the size of the head node which represents free memory.

In the functions that are used by thread such as myMalloc and myFree, first thing I do is to lock our global lock to block other threads from interrupting and affecting data structure. In other words, we do this in order to isolate the lock-owner thread, ensure atomicity, avoid interruptions and data race. Before returning from function and after printing the structure, we unlock the global lock. So, the printed linked-list is the state at the end of the execution of the relevant thread before affection of

other threads. One of the threads can now again obtain a lock and process the data structure just before the returning.

In myMalloc and myFree, we iterate through head with a dummy pointer named ptr. If we can't find a matching node, the ptr will have NULL value and we take actions accordingly. We also keep a previous pointer. If it is equal to NULL, we understand that we apply change to the head of the linked-list and take actions accordingly. We also may update head accordingly or by checking index value. Previous pointer is also useful while merging multiple nodes with "-1" id since we need to check the adjacent nodes in order to merge them into one. All the functions of these 2 thread functions are done as they are asked in the homework file with the usage of a linked list and pointers. We ensure atomicity, isolation, and avoid data race in these thread functions by using a global lock and locking it at the beginning of the function.