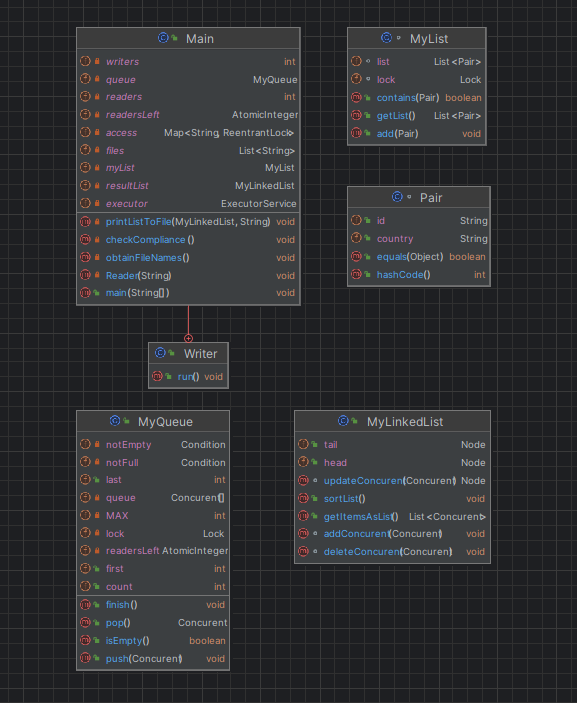
|  |  |  |
| --- | --- | --- |
| **Nr of reader threads** | **Nr of writer threads** | **Time** |
| 4 | 2 | 227 |
| 4 | 4 | 245 |
| 4 | 12 | 260 |

Documentation

As the number of worker threads increases from 2 to 4 to 12 (while keeping the number of reader threads constant at 4), there is a general trend of increasing time. This suggests that increasing the number of worker threads may not necessarily lead to a linear improvement in performance for the given task. The increase in time could be attributed to factors such as increased overhead in thread management, synchronization, or contention for shared resources.

The task or workload being parallelized might not be well-suited for a high number of reader threads. The coordination and synchronization overhead between multiple reader threads may contribute to diminishing returns as the number increases.



* MyLinkedList:

The addConcurent, updateConcurent, and deleteConcurent methods in the MyLinkedList class use explicit locks (lock() and unlock()) to synchronize access to the linked list. This ensures that multiple threads cannot concurrently modify the list, preventing data corruption and race conditions.

The sortList method uses a simple bubble sort algorithm to sort the linked list. While the sorting process itself is not inherently thread-safe, it doesn't involve any external shared resources or dependencies. Therefore, it doesn't require additional synchronization.

* MyQueue:

The MyQueue class uses a ReentrantLock (lock), along with Condition objects (notFull and notEmpty), to manage access to the shared queue. This allows for controlled access to the queue to prevent issues like race conditions.

The push method locks the queue, adds a Concurent object, updates the count, and signals that the queue is not empty. The pop method locks the queue, retrieves a Concurent object, updates the count, and signals that the queue is not full.

The isEmpty method provides a non-blocking check for whether the queue is empty.

The finish method is intended to signal threads waiting on notEmpty and notFull conditions when finishing, allowing them to wake up and exit.

In summary, both classes employ synchronization mechanisms to manage concurrent access to shared resources (linked list or queue). The use of locks and conditions helps prevent data corruption and ensures that the operations on these data structures are thread-safe. It's important to note that the effectiveness of these synchronization mechanisms also depends on how other parts of the program interact with these classes.