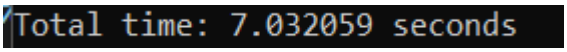
Report1

▼ **hw1a.c**

Multi process program

This code creates an array of 2 million integers, initializes them with random values, and then spawns multiple child processes using `fork()`. Each child process updates a portion of the array by incrementing the value of each element by 1. After all child processes have finished executing, the updated array is printed along with the total time taken for the execution of the program. Finally, the dynamically allocated memory for the array is freed.

Total time: 7.032059 seconds.



▼ **hw1b.c**

Multi Threaded Program With Mutual Exclusion

This code initializes an array of size `ARRAY_SIZE` with random integers and updates its elements by incrementing them using threads. The array is divided into `NUM_THREADS` subarrays of size `SUB_ARRAY_SIZE`, and each thread updates a different subarray. A mutex is used to ensure that only one thread updates an element at a time. The program measures the execution time and prints the updated array and the total execution time.

Total time: 0.127118 seconds.

▼ **hw1c.c**

Multi Threaded Program Without Mutual Exclusion

In this code I made same operations with hw1b but “Without Mutual Exclusion”.

The main difference between this code and the previous one is that we removed the mutex and the `locking/unlocking` mechanism from the thread function. Instead, we allow threads to concurrently update the elements of the array without any synchronization.

This approach is possible because each thread operates on a different portion of the array (i.e., a different subarray), so they don't access the same elements. Therefore, there's no need to protect the array elements with mutual exclusion.
Total time: 0.135051 seconds.

```
Total time: 0.135051 seconds
```

▼ hw1d.c

Single Process Program

This program also updates an array of integers, but it does so without using threads. Instead, it uses a loop to iterate over the array in increments of `SUB_ARRAY_SIZE` and updates each subarray individually.
Total time: 0.002585 seconds.

```
Total time: 0.002585 seconds
```

▼ Conclusion

The results show that the single-process program is the fastest, followed by the multi-threaded program with mutual exclusion. The multi-threaded program without mutual exclusion is only slightly slower than the one with mutual exclusion, but it is still much faster than the multi-process program.

The primary benefit of multi-threaded programs over multi-process programs is that they are more effective and require fewer resources. Race conditions are avoided and data consistency is guaranteed through the usage of mutex in hw1b.c, which limits the number of threads that can update a single element at once. Since the threads in hw1c.c operate on various parts of the array rather than the same components, performance is still good without the use of mutexes.