CS310 Lab 5: Thread Synchronization

October 10, 2023

1. **No synchronization.** Create two threads that increment a global integer counter without any form of synchronization. Conform that the results are incorrect as well as inconsistent. Print the computed value and the expected value (they should disagree.) See the program in fig 26.6 in the textbook.

2. **Lock-based synchronization.** Using `pthread_mutex_lock` and `pthread_mutex_unlock`, add mutual exclusion to the above program. The counter should work correctly now. Verify with the computed value and the expected value (they should now agree.)

3. **Barrier synchronization.** A barrier allows multiple threads to reach the same point, and then continue execution. This can be nicely used for performing multithreaded heapsort. Initialize an array A with 10^8 random integers. A main program creates T threads. Each thread sorts a portion of A. After all threads have finished sorting, the main program merges the sorted portions into the final output. Use `pthread_barrier_init` and `pthread_barrier_wait`.

   - First try it for a small array.
   - Use the `qsort` library function to perform the sorting in each thread.
   - Compare the time taken to sort the same array with a single thread and with T threads. If your computer has 8 cores, put $T = 8$. See the core utilization using the `top` utility (by pressing 1 after running `top`.)