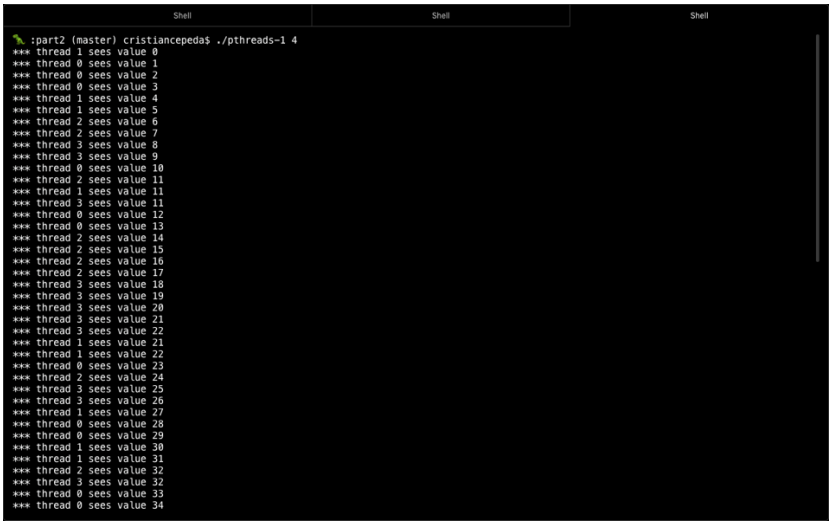
Contributors

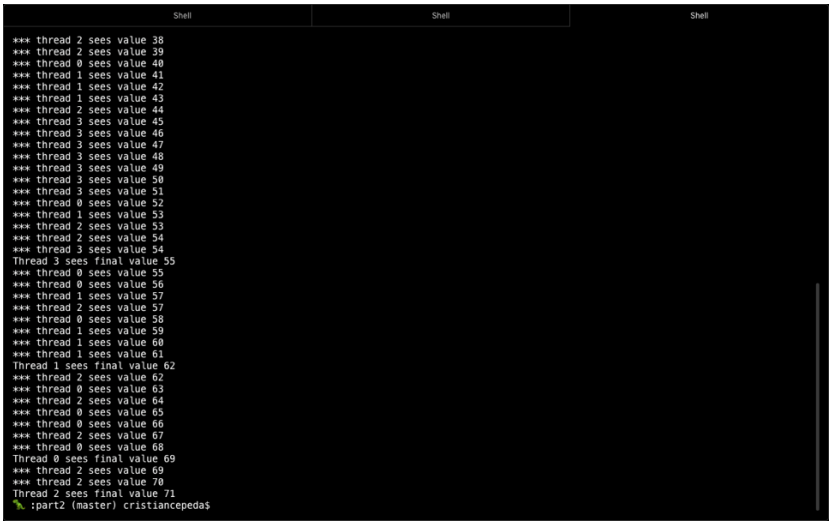
- Cristian Cepeda was the main developer in Part 2 of Assignment-2. However, Marcial Cabrera was jumping in and out since his main focus was on part 1.

Output for Step 1

- Step 1 of Part 2 of the assignment can be seen by running ./pthreads-1 4
    - In this scenario 4 threads start to execute and a clear race condition is noticeable.



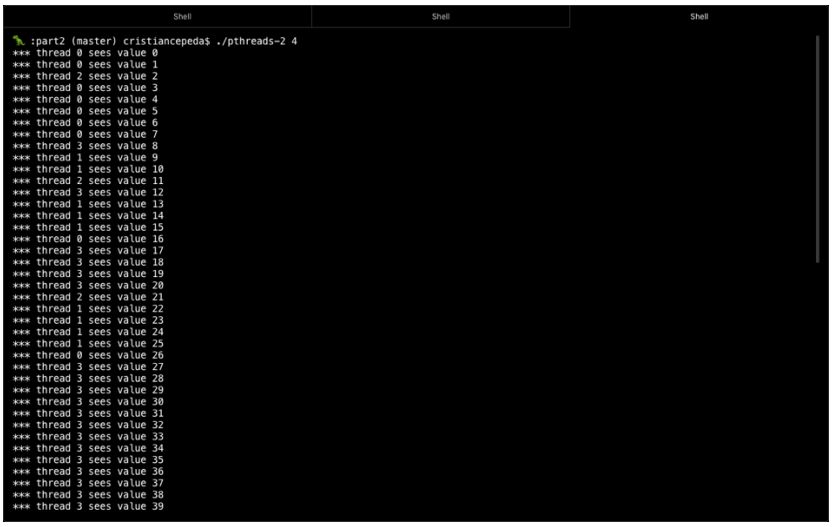
    - The final value in this special case was 71. The real or expected output should have been 80.



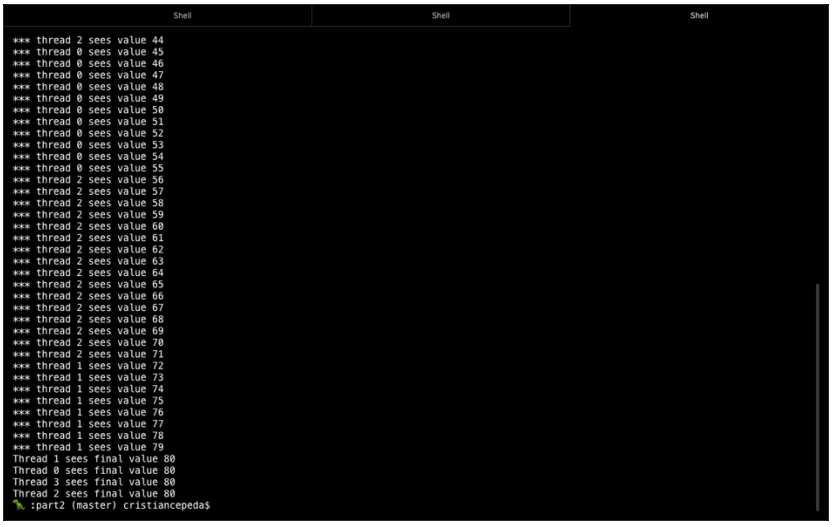
- The reasons why we are not getting the expected output for this section of the project are
    - The critical area of the code doesn't have exclusive access so multiple threads can read and write at the same time causing race conditions.
    - Some threads read the same value and thus have outdated information.

Output for Step 2

- Step 2 of Part 2 of the assignment can be seen by running ./pthreads-2 4
    - In this scenario 4 threads start to execute and we can see that each thread has mutual exclusion.



    - The final value in case is as expected 80. Since we said 4 threads each one adds 20 to the shared variable.



- We got the expected output for this section of part2. This was because mutexes were used to have exclusive access to the shared variable when going into the critical part of the code.
- The last part of this assignment was for each thread to wait for all other threads to finish before they outputted the final seen value. pthread_barrier_t could and should have been used for this part of the code. But since my development environment is a Mac. pthread_barrier_t is not implemented, or it can't run. So, to solve this issue I basically had each thread check in and loop until all threads had checked in before allowing them to continue.