Part 1

1. **Explain the problem and identify evaluation metrics for experiments**

   Is multithreading a faster alternative to sequential operation when finding the number of substrings in a given string, S1? This will be conducted using a sequential program as well as an altered multithreaded program with 3 different inputs for each. Recorded results will be in milliseconds.

2. **Explain your choice of threading libraries**

   I chose to use pthreads as I have not done much threading in the past. Therefore, I had no bias for another library or interest in trying anything new, since this was my first time using pthreads, which was the recommended library.

3. **Explain the design of the experiment and develop programs for evaluation**

   The program used is called "substring_thread.c." This is an adapted version of "substring.c" which can be obtained from the course GitHub.

   Like substring.c, substring_thread.c loops through a given string of N1 characters checking the substring at that spot and beyond from that spot, x, to x+N2 (number of characters in given substring, S2). In this threaded version of substring.c, S1 is split into an equal number of threads to run simultaneously, hopefully speeding up the process.
   I designed my multithreaded adapted program under the assumption that the number of threads used will satisfy the given conditions (Let T be the number of threads):

   - N1%T == 0
   - N1/T > N2

   Experiment:

   (First text file is "test1.txt", second, "hamlet.txt", third, "shakespeare.txt")

   1. Run the first text file in substring_thread.c and collect the time it took to complete under the "1 thread" column.

2. Repeat step 1 four more times to fill in the data table and collect 5 total completion times.
3. Run the first text file in substring_thread.c with NUM_THREADS set to 2 and collect the time it took to complete under the “2 threads” column.
4. Repeat step 3 four more times to fill in the data table and collect 5 total completion times.
5. Run the first text file in substring_thread.c with NUM_THREADS set to 4 and collect the time it took to complete under the “4 threads” column.
6. Repeat step 5 four more times to fill in the data table and collect 5 total completion times.
7. Repeat steps 1-6 for the second text file.

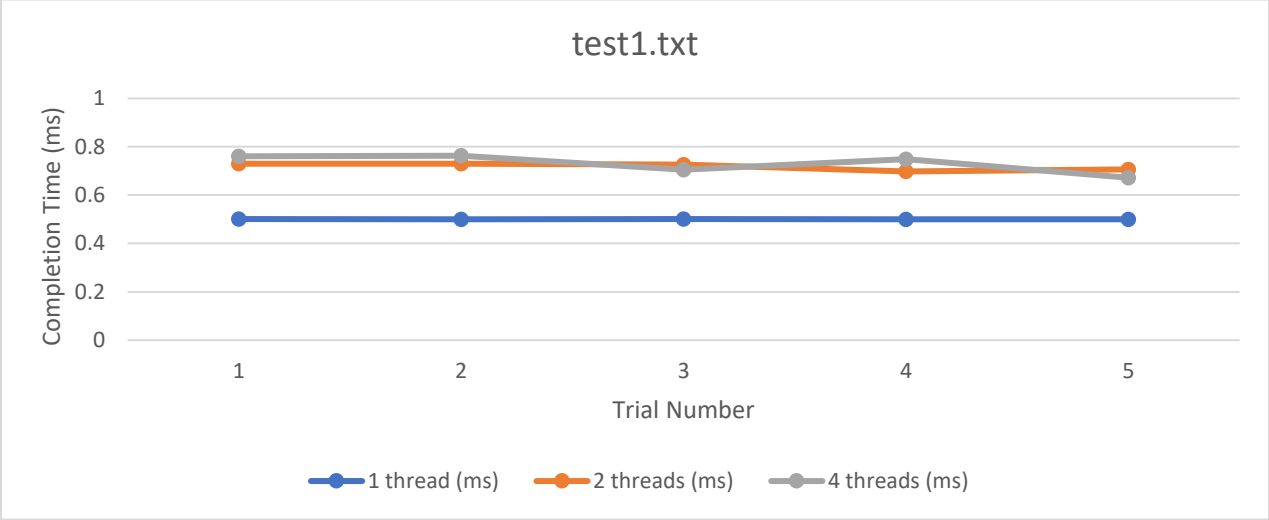
4. Detail your collected experimental results

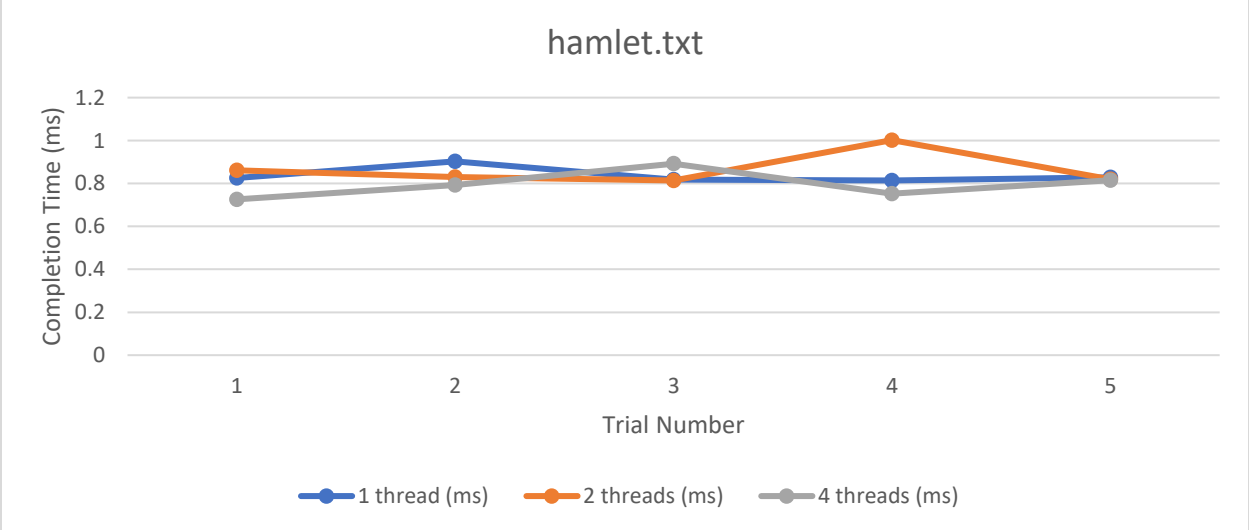
test1.txt (N1 = 21, N2 = 2)			
Run	**1 thread (ms)**	**2 threads (ms)**	**4 threads (ms)**
1	0.501	0.730	0.761
2	0.500	0.730	0.763
3	0.501	0.727	0.705
4	0.500	0.698	0.749
5	0.500	0.707	0.672
Avg	0.500	0.718	0.730

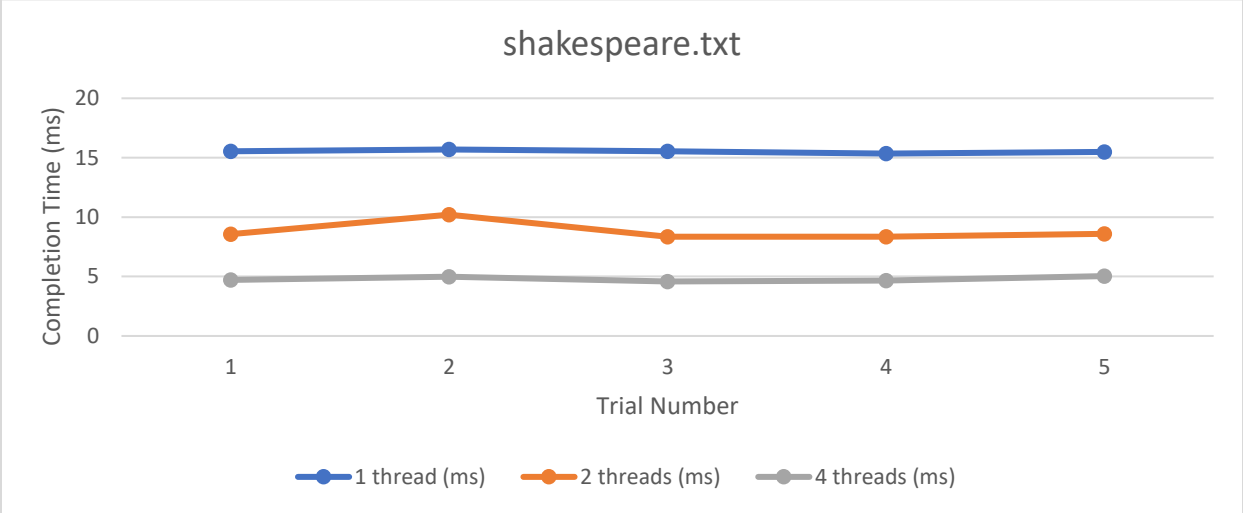
hamlet.txt (N1 = 142,954, N2 = 6)			
Run	**1 thread (ms)**	**2 threads (ms)**	**4 threads (ms)**
1	0.826	0.861	0.726
2	0.903	0.830	0.793
3	0.818	0.814	0.892
4	0.814	1.002	0.752
5	0.828	0.819	0.815
Avg	0.838	0.865	0.796

shakespeare.txt (N1 = 4,358,846, N2 = 3)			
Run	**1 thread (ms)**	**2 threads (ms)**	**4 threads (ms)**
1	15.548	8.555	4.716
2	15.691	10.205	4.984
3	15.550	8.356	4.586
4	15.347	8.353	4.673
5	15.489	8.588	5.038
Avg	15.525	8.811	4.799

5. Analyze, graph, and interpret experimental results and draw conclusions







Analyze, Interpret, Conclude:

From the data I collected, it appears that multithreading is not always the better option when considering the speed of completion for a given program. test1.txt completes faster when run with only 1 thread. Both multithreaded series of runs average the same time, ~7.2ms, while the single threaded series ran at 5.0ms. hamlet.txt could be considered a negligible difference in average completion times between sequential and multithreaded. All threading options completed between a range of ~8-9ms. and shakespeare.txt completes a significant amount faster on a multithreaded program. Each increment of doubling the cores (1 to 2, 2 to 4) seemed to cut the completion time roughly in half, from ~15.5ms to ~9ms, ~9ms to ~5ms.

The difference in performance between a sequential and multithreaded program for varying lengths of strings, S1, could be related to the amount of time it takes to create a number of threads. When S1 is short, then the amount of time used to create threads and join them back together is not worth it. However, when S1 is very long, then the amount of time used to create threads and rejoin them is negligible compared to the time a sequential program takes to go through S1 “alone.” hamlet.txt appears to be the cutoff point of the size of S1 where multithreaded and single-threaded run at similar speeds. It can be assumed that any S1 of size less than hamlet.txt would run faster single-threaded, and any S1 of size greater than hamlet.txt would run faster multithreaded.

For finding the total number of a given substrings in a larger string, multithreading would be a better route if S1 is very large, such as shakespeare.txt. However, a sequential program would be better suited for shorter variants of S1, such as test1.txt.

Part 2

1. **Explain the problem**

   Create a producer-consumer algorithm-based program that reads in a text file one char at a time into a circular queue, and outputs them in the same order.

2. **Explain your choice of threading libraries**

   I chose to use pthreads as I got somewhat familiar with them in Part 1 of this assignment. However, I had not used semaphores with them, so I chose to build upon the knowledge I gained rather than start fresh with a new library.

3. **Explain the design of the experiment and develop programs for evaluation**

   The general design of a producer-consumer algorithm is to have a total of 2 threads. One thread for producing, and one for consuming. The producer thread can produce unless a buffer limit is reached, or until it has nothing more to produce. Likewise, the consumer thread can consume from the buffer unless the buffer is empty.

   In my specific program, the buffer limit is 5, and the buffer is arranged as a circular queue. The producer thread will read from a given file, “message.txt”, one char at a time. The consumer thread will simultaneously read one char at a time from the buffer and print out the char to the terminal.