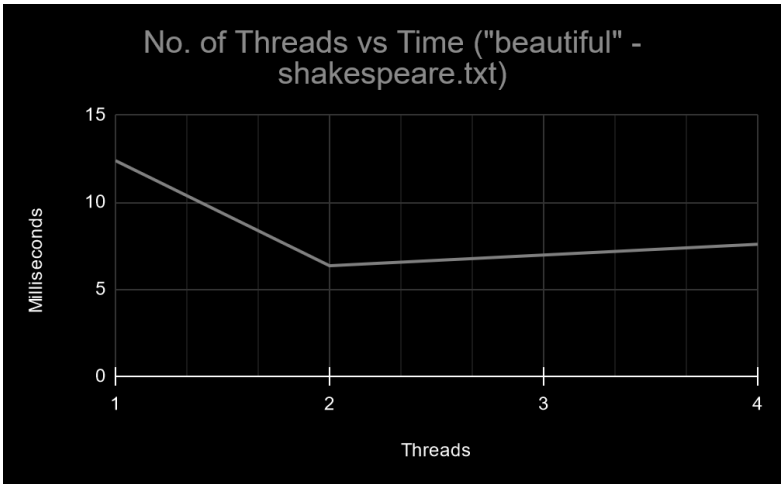
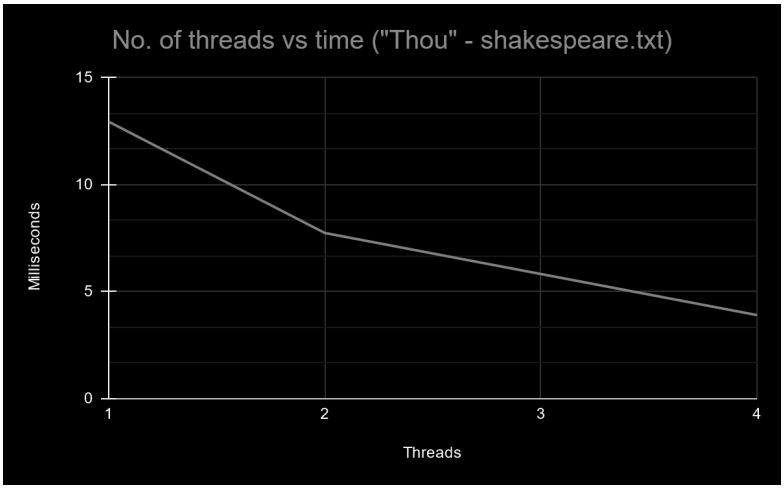
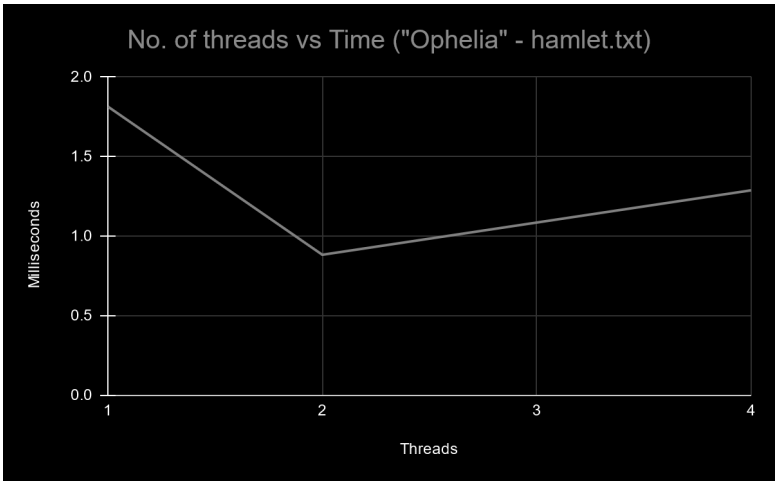
Samarjit Singh Bons
1001623236
CSE 3320-001
Programming Assignment 2
3/28/2021

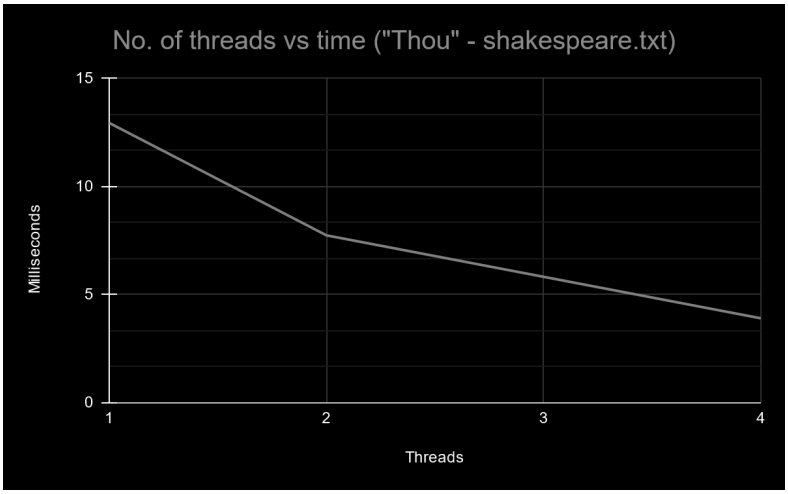
Part 1

1. We are given a text file with two strings, namely s1 and s2. s1 is relatively much larger than s2, and we want to search for the number of occurrences of s2 in s1. The given helper code named “substring.c” provided by the course github repository provides us with a solution implementing an iterative approach to this challenge. The goal is to come up with a solution containing implementing the powerful concept of multi-threading to our program. This will enable us to use multiple cores of CPU simultaneously, hopefully increasing speed and efficiency.
2. I have decided to use the POSIX threads library in C for the assignment. The reason I have used pthreads is because I am the most comfortable with this threading library. **Compile instruction: gcc -pthread -o substring substring.c**
3. The way I implemented pthreads in the assignment is by modifying the num_substring function into a thread function. Instead of going over the entire string s1 iteratively like the sample code, the threads which I am using will each go over their own specific regions with a specified start and an end. The threads are created simultaneously and each one of them looks for substrings in their sector and adds the number found to the grand total of substrings found, which is a global variable, using a mutex. Using a macro definition at the beginning of the program, I can control how many threads I want to execute. In order to evaluate the program, I will collect the time elapsed in milliseconds for the process from start to finish while using different amounts of threads(1,2,4) and compare results.
4. I have used the provided test files “shakespeare.txt” and “hamlet.txt”. I searched for the word “beautiful”(s2) in the string s1. The result was 15 total instances which is a relatively miniscule occurrence compared to the size of the book. The time elapsed was 12.393 ms, 6.355 ms, 7.589 ms for 1,2 and 4 threads respectively. Another word I searched for in the same book was “Thou” which occurs 1462 times. Elapsed time was 12.946 ms, 7.747 ms, 3.91 ms respectively for 1,2 and 4 threads. When I searched for the word “Ophelia” in the book “hamlet.txt”, the words occurred 20 times and the time elapsed was 1.815 ms, 0.884 ms, 1.289 ms respectively for 1,2 and 4 threads. The word “lord” occurs 222 times in “hamlet.txt” and time elapsed was 1.761 ms, 1.201 ms and 1.38 ms for 1,2 and 4 threads respectively.


No. of Threads vs Time ("beautiful" - shakespeare.txt)
Milliseconds
15
10
5
0
1
2
3
4
Threads


No. of threads vs time ("Thou" - shakespeare.txt)
Milliseconds
15
10
5
0
1
2
3
4
Threads


No. of threads vs Time ("Ophelia" - hamlet.txt)
Milliseconds
2.0
1.5
1.0
0.5
0.0
1
2
3
4
Threads



5. A few significant observations can be made after looking at the graphs of the evaluated data. All test cases show significant speed and efficiency when 2 threads are used instead of 1. However, words which occur a significantly large amount of times in a very large file, such as the word “Thou” in shakespeare.txt, enjoy the added benefit of having 4 threads. Others dissimilar words actually even show a penalty in performance compared to using 2 threads. I believe that is because of the added overhead of initializing extra threads which might be more than the time saved using the extra processing power. Note that it did not matter for the word “Thou” because of the significantly larger number of occurrences. This exercise shows us that there are multiple approaches to solving a complex problem and it is not necessary that the most complex solution is the best. We need to understand every problem in detail and tailor our solutions accordingly.

Part 2

1. We are to design a program which uses conditional variables/semaphores to deal with a circular queue which will be manipulated by two concurrent threads, namely producer and consumer. The producer shall read characters from a given file and place them in the queue. At the same time the consumer thread will take the values out from the queue and print them. The challenge here is to make sure that the producer does not overwrite information before it is printed by the consumer and the consumer does not reprint old information by waiting for the producer to add new values.
2. I will use POSIX threads and semaphore.h to design my program because I am the most comfortable with these libraries. **Compile instructions: gcc -pthread -o queue queue.c**
3. The example code “frys.c” provided great assistance with the design of this program. I started by designing a circular queue using an array and a couple iterators and a few helper functions for enqueuing and dequeuing. A producer thread will open a given file if available and read every character in the string provided. Before placing the read

character in the queue, we will wait for semaphore signalling from the consumer thread which dequeues the queue and prints the character. However this activity itself requires a semaphore signal from the producer thread. In order to avoid a deadlock, I prepopulated the queue with one empty element. That seems to eliminate any problems related to deadlocks and the program functions smoothly. Once the producer reaches EOF, the thread ends. Once the producer ends and the queue is empty, consumer ends and the program ends successfully, having printed all the characters from the file.