Assignment Report

Threading with mutex and semaphores.

CSE 3320 – Sec 003

Nicholas Untrecht

1001745062

Professor: Trevor Bakker

March 28, 2021

Part 1 – Substring Threading

Problem

The problem to be solved is creating a substring program that takes any string input and another string to be found in the initial string. While this can be done easily be sequentially searching the large string, it can be done faster with threading. Each thread will search a designated area of the large string, assigned by the number of threads available for the program.

Evaluation can be done easily by timing how long each search takes and comparing that to the initial non-threaded, sequential search.

Threading Library

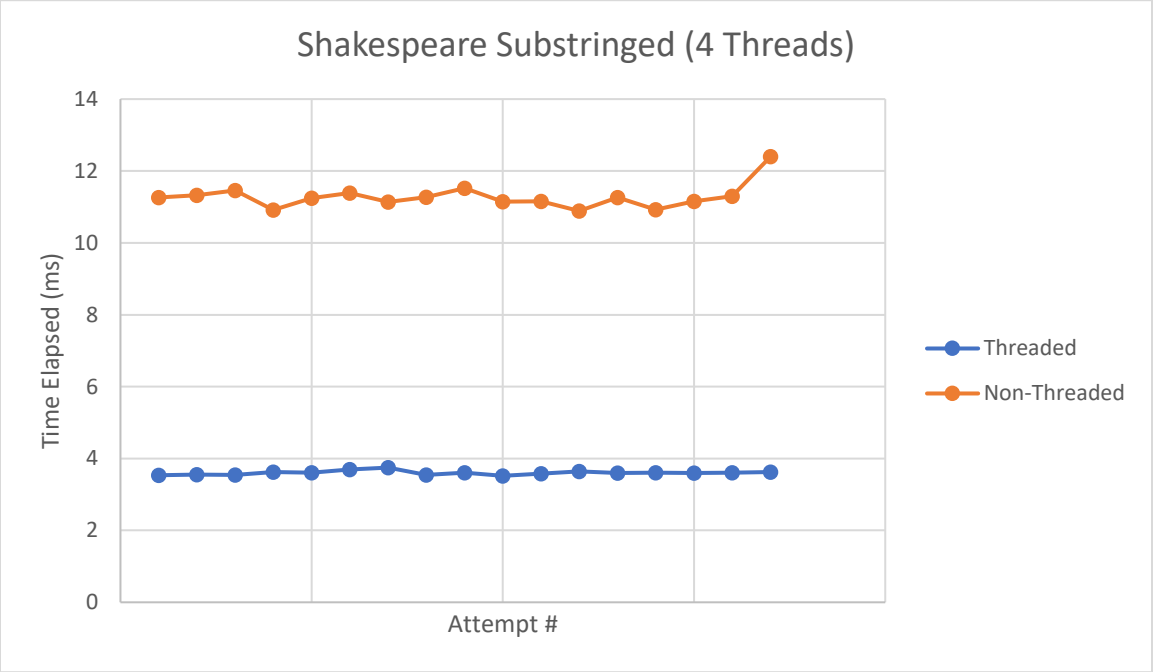
My choice for a threading library was POSIX Threads (pthreads). It is easily integrated into C programming, all that's needed is a `#include` statement and a link when compiling. Mutexes are available and integrated into the library (which are necessary for preserving the substring count value). GDB also supports threading debugging so it is viable to watch the program execute with each thread.

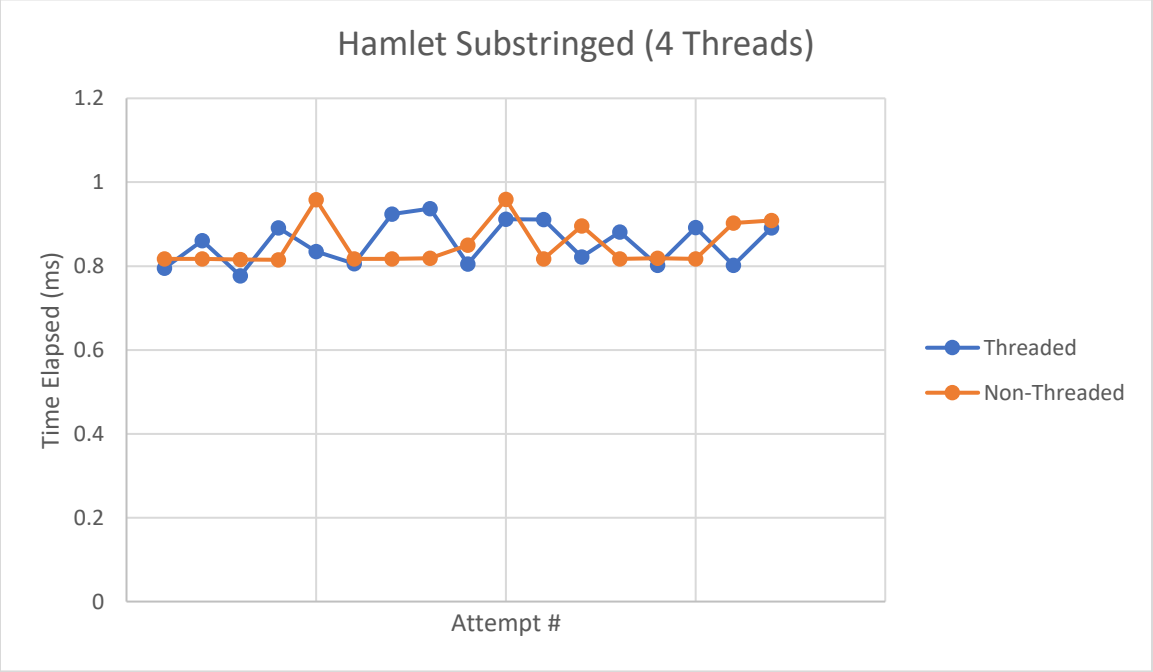
Experiment

The experiment will consist of gathering execution timings for both non-threaded and threaded programs. This is done by pulling the time of day before the substring algorithm starts and finding the difference between the time of day once the threads join back together. Each program will run with each document 20 times, and this data will be printed out on execution and inputted into spreadsheets for graphing results.

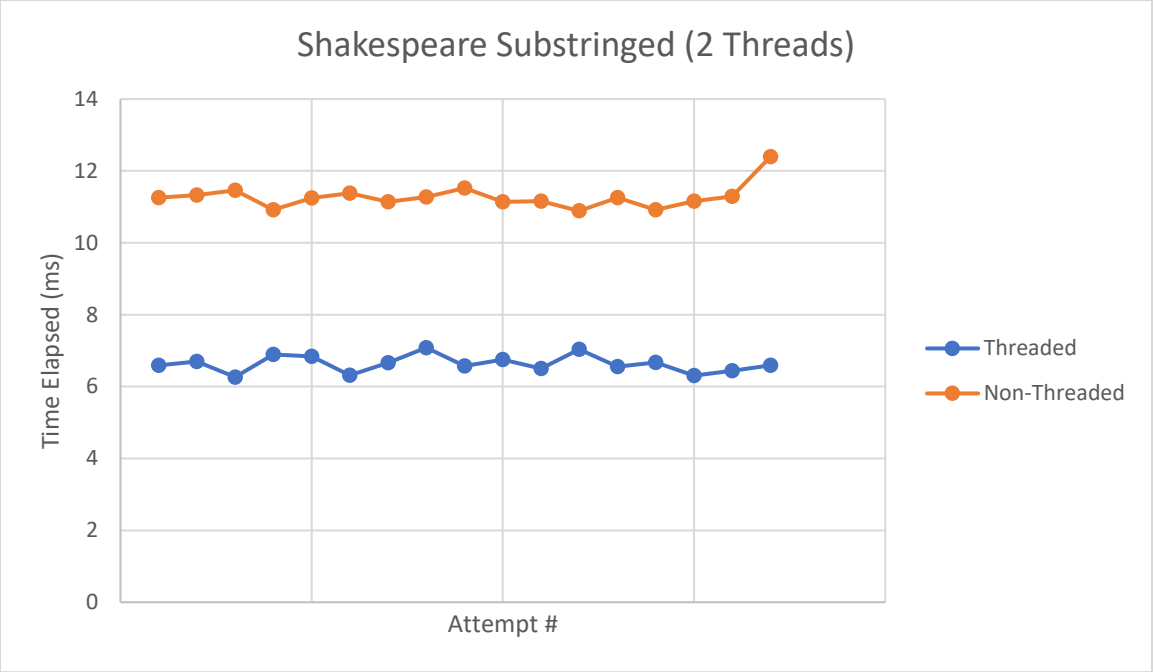
The time values will be tested for 2 text documents: shakespeare.txt and hamlet.txt. Thread values of 4, 2, and 1 will be tested with the threaded version and compared to the non-threaded version.

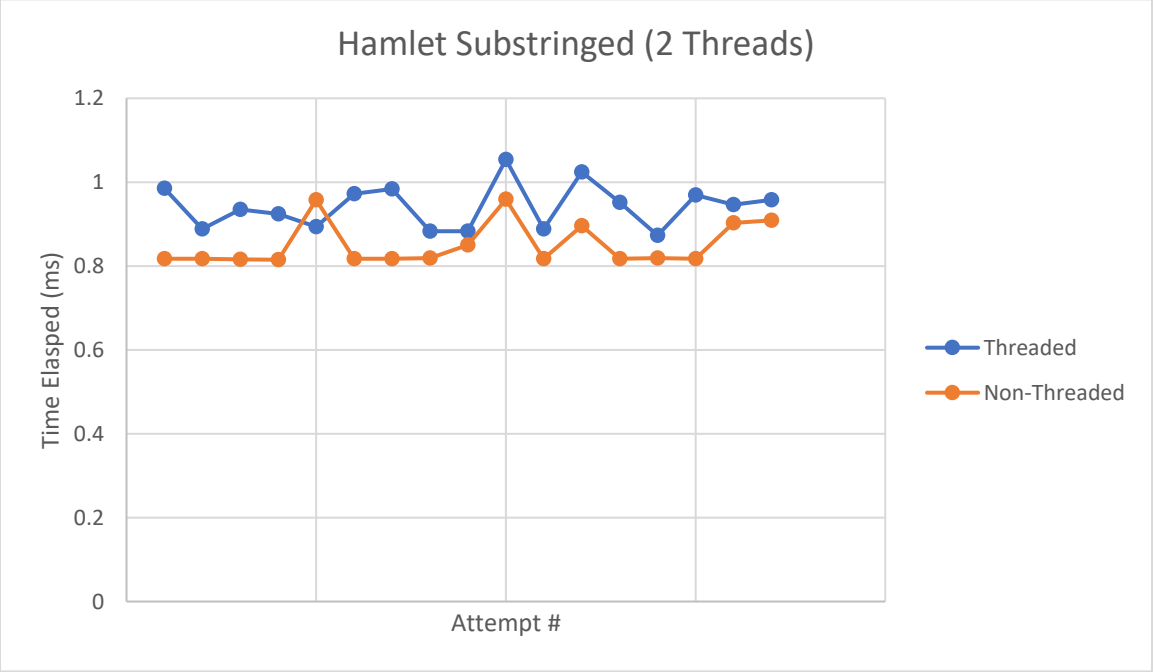
4 Threads


Shakespeare Substringed (4 Threads)
Time Elapsed (ms)
14
12
10
8
6
4
2
0
Attempt #
Threaded
Non-Threaded

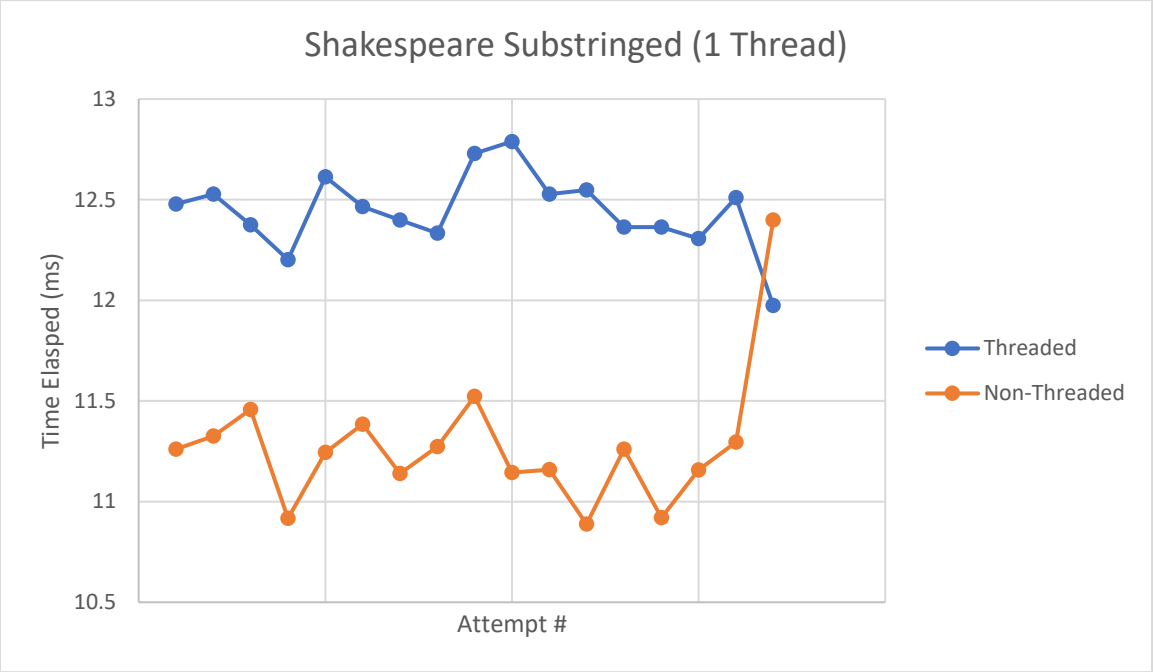

Hamlet Substringed (4 Threads)
Time Elapsed (ms)
1.2
1
0.8
0.6
0.4
0.2
0
Attempt #
Threaded
Non-Threaded

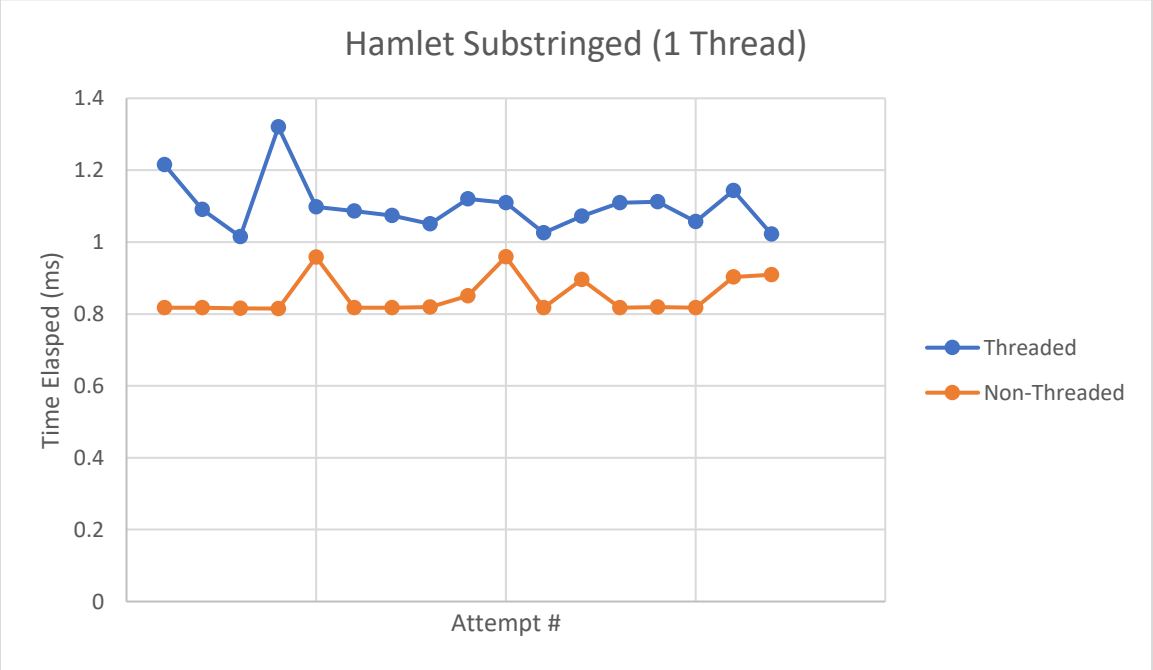
2 Threads


Shakespeare Substringed (2 Threads)
Time Elapsed (ms)
14
12
10
8
6
4
2
0
Attempt #
Threaded
Non-Threaded


Hamlet Substringed (2 Threads)
Time Elapsed (ms)
1.2
1
0.8
0.6
0.4
0.2
0
Attempt #
Threaded
Non-Threaded

1 Thread


Shakespeare Substringed (1 Thread)
13
12.5
12
11.5
11
10.5
Time Elasped (ms)
Attempt #
Threaded
Non-Threaded


Hamlet Substringed (1 Thread)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Time Elasped (ms)
Attempt #
Threaded
Non-Threaded

Analysis

With all of these tests, there is the important concept of **context switching** that will factor into all comparisons. The main factor to test is to see if the *cost* of switching threads can outweigh the normal cost for searching sequentially.

Firstly, the Shakespeare file: when used with 4 and 2 threads, the performance increase is notable, and somewhat factored by the number of threads used. The 2-threads version is roughly 2x faster and the 4-threads version is roughly 3x faster. With an occasional outlier, the 4-threads performance here is really consistent, with no deviation more than 0.2 ms. On the other hand, the 2-threads version has more inconsistency, deviating at most 0.7 ms. When discussing the 1-thread version, the performance is higher than the normal sequential. This is because it is the normal sequential version with added context switching and thread code that adds roughly another 1 ms to execution.

Next, the Hamlet file: the 4-threads version is roughly comparable to the non-threaded version, but still not ideal. While the 4-threads version does have the *lowest possible* time (0.777 ms), the non-threaded version actually has more consistent performance (0.82 ms often). 2-threads shows the context switching threshold here. The 2-threads version has an average performance difference of 0.1 ms compared to the sequential; this is a very small difference but with the consistency of the non-threaded version, it is clear who wins here. With the 1-thread version, similar to the Shakespeare 1 thread version, it has a noticeable decrement in performance (0.3 ms worse) compared to the non-threaded version.

*spreadsheet for data is included with report.

Compiling and Implementation

Compilation: Makefile included, `‘make thread’` for default program.

-lpthread argument linking used in makefile for libraries.

Implementation: substring.c and Makefile will be included with this report.

Part 2 – Producer-Consumer

Problem

The problem to be solved here is a producer-consumer algorithm regarding text documents. The producer will write characters from a text file into a circular queue (one by one), and the consumer will write them out of the queue.

Threading Library

All previous reasons stand for choice of threading here (integration in C, GDB integration). Semaphore (or condition variables) are included in the pthread library in C, but a separate `#include` statement is needed for use.

Compiling and Implementation

Compilation: Makefile included, `‘make queue’` for default program.

-lpthread argument linking used in makefile for libraries.

Implementation: consumer.c and Makefile will be included with this report.