Hoang Ho
ID 1001654608
CSE 3320-003
Due 03/28/2021 11:59 PM CST

Programming Assignment 2: Threads – Report

Part 1

A text file with two lines is passed in as a command line parameter when the program part1.c is executed. The program reads in each string in each line as s1 and s2 respectively. The amount that substring s2 occurs in s1 is counted using a multithreaded program. To implement multithreading, the POSIX thread (pthread) library is used, since it is already included in the VM being utilized without needing to install additional libraries. The number of substrings in s1, the elapsed time it takes to count all substrings within s1 in milliseconds, and the number of threads being used is taken into consideration and measured when conducting the experiment.

The experiment that finds the amount of time spent on finding the number of substrings with varying number of threads is conducted. This involves adjusting the number of threads in part1.c to be used with 4 vCPUs on the VM. This experiment uses part1.c for evaluation and additionally takes into consideration the size of each test text file used as command line input for the program. For the tests, hamlet.txt and shakespeare.txt is used. By default, the number of threads within the program to be used is defined as 4 threads. For this experiment, 0, 1, 2, and 4 threads will be tested. For 0 threads, the program substring.c provided by CSE3320 github repository will be used instead of the multithreaded part1.c.

Compilation:

- 1 thread: gcc part1.c -o part1 -lpthread -D NUM_THREADS=1
- 2 threads: gcc part1.c -o part1 -lpthread -D NUM_THREADS=2
- 4 threads: gcc part1.c -o part1 -lpthread

Results

Table 1: hamlet.txt (143 KB)

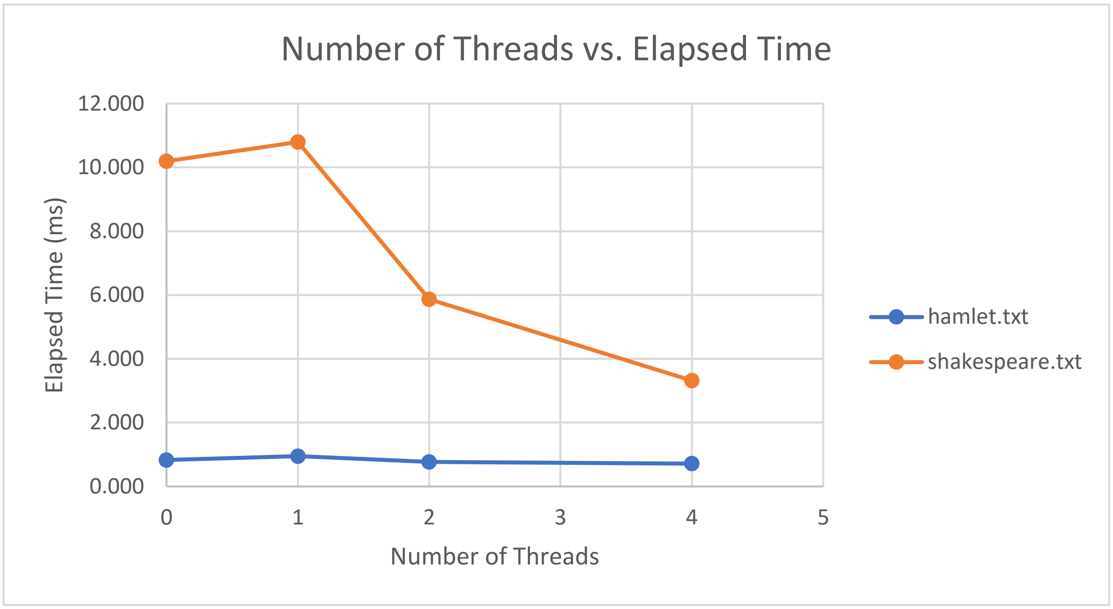
Number of Threads	Number of Substrings	Elapsed Time (ms)
0	85	0.830
1	85	0.951
2	85	0.773
4	85	0.715

Table 2: shakespeare.txt (4400 KB)

Number of Threads	Number of Substrings	Elapsed Time (ms)
0	153	10.197
1	153	10.796
2	153	5.865
4	153	3.319

Hoang Ho
ID 1001654608
CSE 3320-003
Due 03/28/2021 11:59 PM CST

Figure 1: Results for Each Text File



As expected, each thread produced the same number of substrings that was found searching through each text file. For hamlet.txt, “Hamlet” was searched and for shakespeare.txt “Romeo” was searched.

Analysis and Conclusions

For both text files, using 1 thread is worse than using 0 threads based on elapsed time. To see any improvements in performance time, at least 2 threads must be used.

As shown by the data and graphs, the time spent searching for the substring for shakespeare.txt dramatically decreased with each increase in number of threads. The elapsed time for hamlet.txt only slightly decreases with each increasing number of threads. From 1 thread to 2 threads, the time halved for shakespeare.txt. For hamlet.txt, the time from 1 thread to 2 threads to 4 threads did not change as drastically compared to shakespeare.txt, especially from 2 threads to 4 threads. This may be since shakespeare.txt is a much larger file with 4400 KB than hamlet.txt with 143 KB. When using threads, the work to be done is split up and divided based on the number of threads. Having more threads on a smaller file may not improve performance time as drastically as a larger file, because the size of each split-up work or partition is on a smaller scale on a small file. It would be best to use less threads on a smaller file compared to a larger file. On a larger file, each partition would be a size moderate or large enough to see a difference with each thread working on its portion.

Hoang Ho
ID 1001654608
CSE 3320-003
Due 03/28/2021 11:59 PM CST

Part 2

A producer-consumer algorithm is implemented into a program called part2.c utilizing two threads: one producer and one consumer. A test file called "message.txt" is created with "Hello World!" within it and is stored in the "tests/" directory. The POSIX thread (pthread) library is used for the threading in this program since it is already provided in the VM without having to install additional libraries. The producer writes each character from this text file one by one sequentially to a buffer queue of 5 characters. At the same time, the consumer reads from the buffer queue as each character is written to it and tells the producer through its consumer semaphore that it is ready to read more characters. The producer also tells the consumer when it has written something to the queue for the consumer to read with its producer semaphore. Both producer and consumer keep track of their positions with prodQueuePos and consQueuePos variables making sure they do not collide when doing their operations. The output prints one character at a time on a line as it is read from the consumer, so that the program is shown working. The producer makes sure to tell the consumer when it has read all of the characters it can read from the file and ends the program when all characters are printed out. Some evaluation metrics to see if the program is functioning properly is watching the characters as they are printed in the correct order in real time, and also not stuck infinitely running even after all characters are outputted.

Compilation: `gcc part2.c -o part2 -lpthread`