Report

Part One:

Part one deals with writing parallel program using threading library to find number of substrings in a given string. The goal is to partition given string for threads to concurrently search for matching substring and print out total number of matches.

I approached this problem by first finding the number of substrings in the string and printing out the total matches with the time taken for the compiler to find the matches without any use of thread. My program consists of 3 main functions. I have a function, int readf, that reads the string from the file and stores it to s1 and stores the substring to s2. Num_function finds the total substring in the string and the main function is where the total is printed.

Then, I added pthread threading library and set up the number of processors in my VM to four. The reason I chose pthread library was because I had prior experience using pthread and I find it to be the most efficient for parallel processing. Also, because the VM we're using comes with pthread, it was convenient to use pthread library. Next, I split the work between processors to find the substrings. I set up the number of threads in my program to one and compiled it. Then I gradually increased the number of threads in my program to 4 so see how it affects the time the complier takes to find the total substrings.

As I noted down the time elapsed with 1 thread, 2 threads and 4 threads, I noticed that with the increase in number of threads, the elapsed time was decreasing. I used two texts, shakespeare.txt and hamlet.txt, for this experiment and the data are as follows:

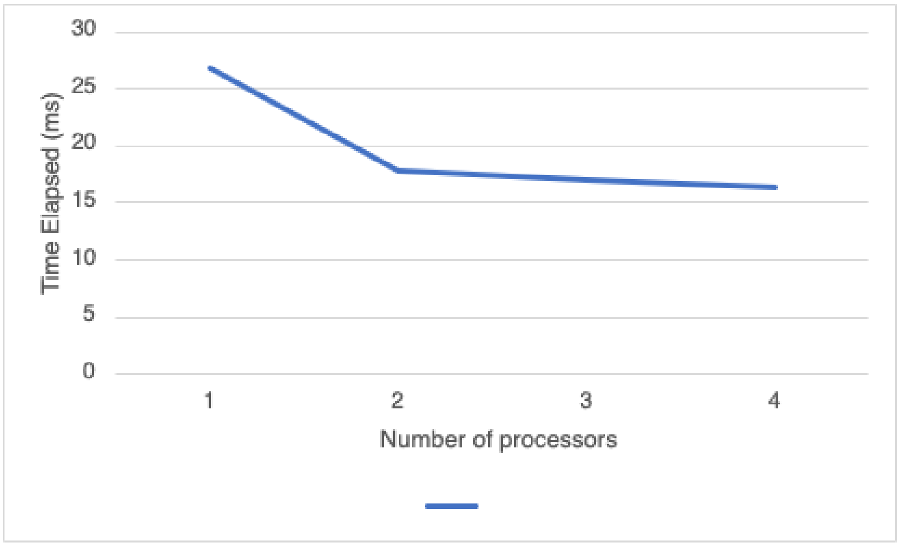
To compile:

```
gcc -o thread thread.c -lpthread

./thread hamlet.txt
```

Shakespeare.txt

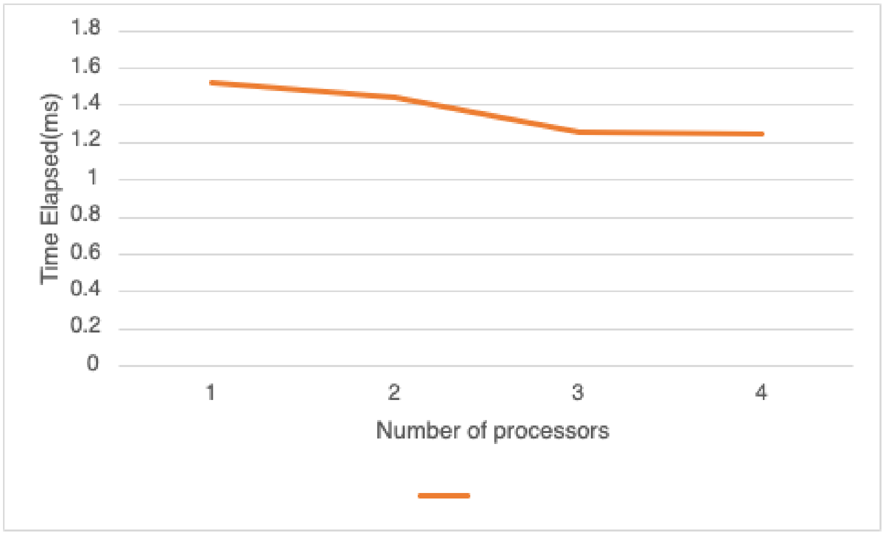
Number of threads	Elapsed time(ms)
1	26.923
2	17.897
4	16.373



Hamlet.txt

Number of threads Elapsed time(ms)

1	1.527
2	1.448
4	1.243



As we can observe from the data and the graphs, increase in number of threads led to decrease in elapsed time. Multithreading allows multiple processors to work on a problem concurrently, so as we increase the number of threads, there are more processors working on the problem at the same time which offers speedup.

Part two:

Part two implements condition variables to implement producer-consumer algorithm. Producer reads character one by one from file “message.txt” and writes the characters to a queue of size 5. Consumer reads sequentially from the queue and prints them in the same order. The problem applies use of semaphores, used for process synchronization. I used pthread library because it allows single thread to wait on another termination.

I used three functions for the problem. Producer function opens the file and reads every character in it one by one and writes it in a queue. I used semaphore to control access to the queue so that chars are stored in it only when there is an empty slot in the queue of size 5. I used mutex to guard the queue. Consumer function reads from the queue and prints the characters in the queue sequentially. It has semaphores wait and post that waits till something is produced and signals the producer. Mutex guards the buffer again. And finally, the main function initializes the threads and semaphores and creates and joins the producer thread and consumer thread.

To compile:

```
gcc -o proCon proCon.c -lpthread

./proCon
```