Blake Cruz

Code Report

My code was based on the template provided in the project library. This means that I was using “#include <pthread.h>” to create pthreads for the multithreading process. We will have 10 threads as per the assignment requirements and we will need to view and process a text file. This report will primarily go through the sections I created, and in my code I have also included extensive comments for further clarification of my efforts.

A majority of the logic I created is located within the num\_substring method. This method takes in an integer (t) as a parameter which represents the thread id which will be responsible for checking the partition of s1 assigned to a specific thread. We keep track of the amount of substring matches with the “currentCount” We will then calculate the length of the amount allocated of s1 and store this in “subLength”. This calculation is simply dividing the length of the substring by the number of threads to get the partion for each thread. We find out the specific thread’s spot in the s1 string by assigning it a “begin” (multipled by “t”) and an “end” (begin + subLength) and the thread will only check this portion of the s1 string.

Then we will iterate through a nested loop where we will check each character from s2 against the current characters of the current s1 thread. If there is a match then the currentCount gets increased by 1, if not then keep searching the designated partition. Once the loop finishes return the count.

In the callSubstringThread method I elected to include print statements that include the thread id and how many matches they found.

Finally in the main method I have not changed much from the template but I have included the final calculation for the count using the totalNum variable. We finally print this value to the console.

Compared to the linear implementation this code does run faster when the strings.txt file is considerably large. Spreading the workload across 10 threads decreases the time to search at the expense of computer resources. Using the default strings.txt file I managed to get a time of .001561 seconds of CPU time. I did start my time after the file was read but before any thread work begins.

Below are my example outputs in the Linux console.

OUTPUT EXAMPLES:

Provided strings.txt

A screenshot of a computer program

Description automatically generated

Modified strings.txt

A screenshot of a computer program

Description automatically generated