Batuhan İşcan
11863132364

CMPE 382 Homework 2 Report

In this assignment, I used the pthread_t and sem_t to write a program that counts the number of prime numbers in a text file.

First, since we do not necessarily know the names of the files or the amount of files, we use the first argument (besides the program name) as the directory in which the text files are read from. A DIR pointer is created with the inclusion of dirent.h and a dirent struct is made to be able to iterate over the entries in the directory.

Secondly, the names of the text files are read and concatanated to their directory and saved into a text file for later. This file is closed and reopened with a “r” access instead of “w” that was used to write it.

An indefinite while loop is established and the break condition is also implemented, it being the “files” reaching the end of the file. In order to pass both the thread id and the text file that is going to be read, I made a struct called thread_data and saved these information that are to be passed to the functions by the pthread_create in this section.

There are semaphores inside the while loop that were initialized at the start of the main function. I have used 3 different semaphores which are “mutex” that has a capacity of up to the thread_count that is the second taken console argument and used to simulate how many threads can go into the function at the same time, “limit” which limits each thread to read the “files” file one by one and the “write_lock” which locks the output being written as there might be racing situations in there which might cause output errors.

The filename and semaphores are adjusted as necessary and the threads and the thread_data are passed into the parallelised function. In this function, the thread_data that was taken as a void pointer is recast into the struct thread_data and its fields are taken to be used. The file that the data uses is taken as the relative path from the thread_data and opened as a “r” access file. Then, the values are read and sent to another function that calculate the number being prime or not. If that functioned return value is 1, the prime count is incremented and the final result is printed after the file reaches its end.

The file that checks the number being prime first checks the number being 1, 2 or 3, then checks if the number is a multiple of 2 or 3 by the modulus operation. This provides an early return for more time efficiency. Afterwards, the first few prime numbers are compared to avoid checking every number whereas holding an integer array of around 15 elements should immensely ease the brute force method.

Besides the aforementioned comparisons, the current array element’s squared value is compared to the value. If the squared number is higher and a prime divident has not been found yet, this number needs to be a prime number. This comparison also provides an early return. Afterwards, a for loop is used to check every odd number to divide the value with until the squared value is greater than the value.

Below, there is the performance table for the 20-30 files, each with 1000 random integers:

Batuhan İşcan
11863132364

THREADS	REAL	USER	SYS
1	0,010s	0,007s	0,000s
2	0,010s	0,013s	0,000s
3	0,010s	0,015s	0,000
4	0,013s	0,014s	0,014s
6	0,008s	0,012s	0,000s
8	0,008s	0,014s	0,000s
12	0,007s	0,010s	0,000s
16	0,007s	0,010s	0,000s

In retrospection, the number of integers could be increased as the discrepencies in the performance would be more apparent then. This data was taken as an average of 20 runs for each thread count and avoided outliers that other system processes might have caused. The similarity in the time for lower thread counts might imply that the semaphore limitations and each thread needing to work for much more time than the higher thread counts has affected the outcome. As there are only a few threads, the loss of time that occurs when both threads get stuck at the write_lock is much more significant than the higher thread counts.

It can also be said that although the higher threaded runs are more likely to get stuck at the limit or write_lock semaphores, the amount of threads make up for the time loss.