**CSE 3033 PROJECT 3 REPORT**

150121025 Ahmet Abdullah Gültekin

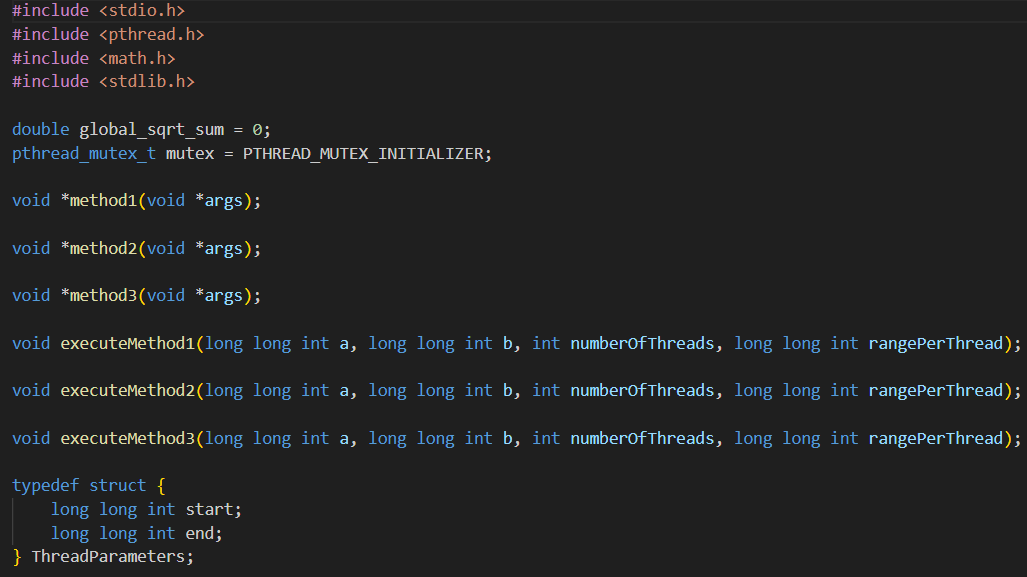
150120035 Ömer Deligöz

150120004 Melik Özdemir

**DESCRIPTION:**

In this project we are expected to write a program that uses threads and synchronization to find the sum of the square root of numbers in a range given by the user.

**CODE:**

****

In this part, there are some variables and name of the functions. We have sqrt\_sum variable which has double type. The other variable is mutex which has pthread\_mutex\_t type is from pthread.h library. Additionally we have 6 function which ahve void type. Finally, we have one struct that includes 2 variable which have long long int data type.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

In this part, we calculate the square root of numbers in a given range and updates a global sum. This function is designed to be executed as a separate thread using pthread. It takes a ThreadParameters struct as argument, which contains the range of numbers to process. The function calculates the square root of each number in the range and adds it to the global\_sqrt\_sum variable.

args A pointer to a ThreadParameters struct specifying the start and end of the range. This function does not return a value.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

This part represents a method that calculates the sum of square roots in a specified range using multiple threads. This method is designed to be run in a separate thread to perform the calculation concurrently with other threads. It takes a pointer to a `ThreadParameters` struct as a parameter, which specifies the start and end values of the range for the calculation. The calculated sum is added to the `global\_sqrt\_sum` variable, using a mutex to ensure thread safety. Args Pointer to a `ThreadParameters` struct containing the start and end values of the range. This function does not return a value.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

This function calculates the sum of square roots for a given range of numbers using multiple threads. This function is used to calculate the sum of square roots for a range of numbers using multiple threads. It receives a struct of type `ThreadParameters` as an argument, which contains the start and end values for the range. Each thread calculates the square root of each number in the range and adds it to a local\_sum variable. The local\_sum is then added to the global\_sqrt\_sum using a mutex lock to ensure thread-safe access. Args a void pointer to a struct of type `ThreadParameters` that defines the range of numbers to calculate the sum of square roots. This method does not retun a value.

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

This method executes METHOD 3 with multiple threads. This function divides the range between 'a' and 'b' into 'numberOfThreads' parts and creates a thread for each part to calculate the sum of square roots. Each thread receives a 'start' and 'end' value specifying the range of values to calculate the sum of square roots. The main thread waits for all threads to finish before printing the final result. “a” is the lower bound of the range. “b” is upper bound of the range. “numberOfThreads” is the number of the threads to create. “rangePerThread” is the size of the range to assign to each thread.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

This method executes METHOD 2 with parallel threads. This function executes method 2 with parallel threads. It divides the range from 'a' to 'b' into'numberOfThreads' sub-ranges, and assigns each sub-range to a separate thread. Each thread calculates the sum of square roots within its assigned sub-range. The final result is the sum of square roots from all threads. . “a” is the lower bound of the range. “b” is upper bound of the range. “numberOfThreads” is the number of the threads to create. “rangePerThread” is the size of the range to assign to each thread.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

This method executes METHOD 1 using multiple threads. This function splits the range [a, b] into smaller ranges and assigns each range to a separate thread. The number of threads is determined by the parameter numberOfThreads. Each thread calculates the sum of square roots within its assigned range and updates the global\_sqrt\_sum variable. After all threads have finished, the function prints the final result. “a” is the lower bound of the range. “b” is upper bound of the range. “numberOfThreads” is the number of the threads to create. “rangePerThread” is the size of the range to assign to each thread.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

This is main function. First of all,we check if the program is called with exactly five command-line arguments. If not, it prints a usage message indicating how to properly use the program and exits with a return code of 1, signaling an error. Then, we convert the command-line arguments (argv) to appropriate data types. “atoll” is used to convert strings to long long int, and “atoi” is used to convert strings to int. The converted values are stored in variables a, b, c, and d, respectively. Finally, we use a switch statement based on the value of methodNumber to determine which method to execute. Depending on the value of d, it calls one of the functions executeMethod1, executeMethod2, or executeMethod3 with the provided arguments. If the value of d does not match any case, it prints an error message and returns with an exit code of 1.

**TABLES:**

**METHOD 1 RESULTS:**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**METHOD 2 RESULTS:**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**METHOD 3 RESULTS:**

**metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**QUESTIONS AND ANSWERS:**

**1-Which method(s) provide the correct result and why?**

Methods 2 and 3 always give correct results because they use mutex when changing global variables. However, method 1 gives incorrect results when the number of threads is more than 1.

**2-Among the method(s) providing the correct result, which method is the fastest?**

Method 3 is the fastest method. Because in method 3 sum value kept locally and added to the global value at the end so all threads can run simultaneously. However in method two, threads constantly edit the global variable and use mutex for this, so only one thread can run at a time and all other threads are in busy waiting state.

**3-Among the method(s) providing the correct result, does increasing the number of threads always result in smaller total time? Discuss this considering the number of CPU cores available in your computer (in Linux, lscpu command provides the number of CPU cores available in your computer)**

Accordding to our results that are obtained from a computer which has 6 cores and 12 threads, the argument that is provided by question is true for method 3. But ıf we consider method 2, we can not say that increasing the number of threads always result in smaller time. Because it varies for method 2.

**metin, ekran görüntüsü, yazı tipi, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**4-Are there any differences in user time/system time ratio of the processes as the number of threads increases? What could be the cause of these differences?**

In method 2, the user time/system time ratio always decreases. In method 3, this ratio always increases. Because of this result have some reasons:

Method 2 writes the results to global variable one by one. So system time increases because of context switch times.

Method 3 writes the results from local variable to global variable at the end of the calculation. Since threads can run at the same time, user time increases, so the rate increases.

As a result, we can say that writing the results to global variable one by one takes more time from writing the results from local variable to global variable at the end of the calculation. It is about the context switch time.