Marmara University
Faculty of Engineering

CSE 3033
Operating Systems

Project 3

Instructor: Ali Haydar ÖZER Date:03.01.2024

	Department	Student Id Number	Name & Surname
1	CSE	150120012	Kadir BAT
2	CSE	150120055	Muhammed Talha KARAGÜL
3	CSE	150121021	Feyzullah ASILLIOĞLU

CONTENTS

Abstract

The provided code implements three different methods for calculating the square root sum of a range of numbers concurrently using multiple threads. The methods differ in how they handle synchronization:

- Method 1: Each thread independently updates the global sum without synchronization.
    - Expected Behavior: For the same input, we expect the results (sum) to be different when the number of threads changes. This is because, without mutex synchronization, threads do not work concurrently. Consequently, Method 1 is prone to race conditions, and the absence of synchronization may lead to incorrect results.
- Method 2: Each thread acquires a mutex before updating the global sum to ensure mutual exclusion.
- Method 3: Each thread calculates a local sum, and after the local sum is computed, it acquires the mutex to update the global sum.

Questions:

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

Method 2 and Method 3 are likely to provide correct results because they use a mutex to synchronize access to the global sum. This ensures that multiple threads do not concurrently update the global variable, avoiding race conditions.

Method 1 is prone to race conditions, and it may produce incorrect results. Multiple threads updating the global variable simultaneously can lead to data corruption.

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

Method 3 is by far the fastest method. Method 2 is quite slow. Although method 2 and method 3 give the same result, there is a speed difference between them. Method 1 is faster than method 2 but gives wrong results.

3. Does increasing the number of threads always result in a smaller total time?

Not necessarily. The speedup achieved by increasing the number of threads depends on the nature of the workload and the available CPU cores.

If the workload is not parallelizable or if the number of threads exceeds the number of available CPU cores, increasing the number of threads may not result in a smaller total time.

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?

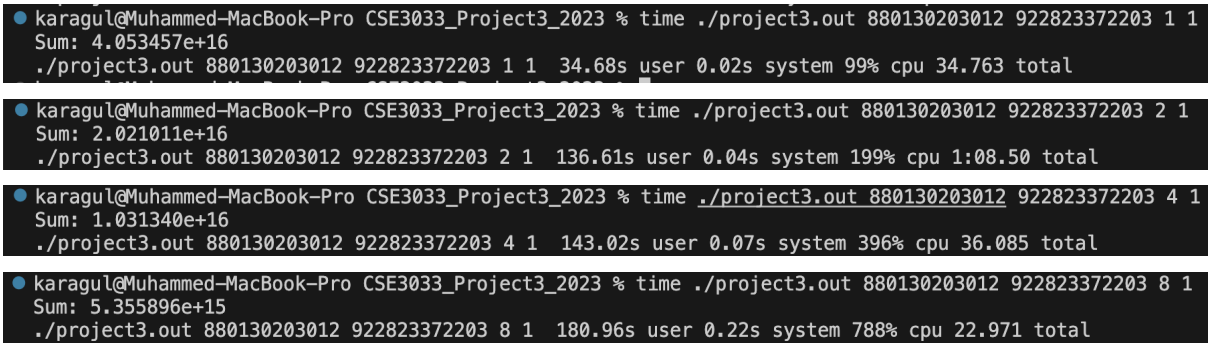
As the number of threads increases, there is an increase in both user time, indicating more time spent in user-level code, and system time, signifying an escalation in kernel-level operations.

Increasing the number of threads may lead to more CPU contention, affecting the ratio. Mutex operations and synchronization mechanisms can contribute to kernel-level activities, impacting the ratio.

Outputs

We ran the program on 3 different computers. One of them has Linux and the other has a Mac operating system. On the 3rd computer, we installed Linux with a virtual machine and tried it. We got the results the fastest on Mac, so we filled the tables with the data we got from Mac. On the other two computers we had to wait very long times for methods 1 and 2, so we put some of the data we ran on Linux at the bottom of the report.

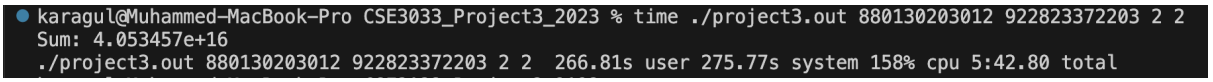
Method 1

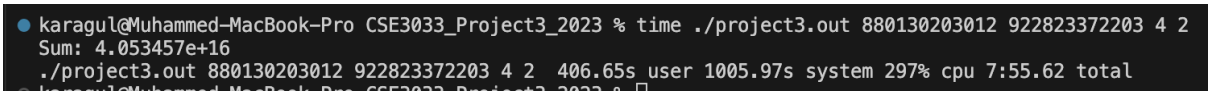


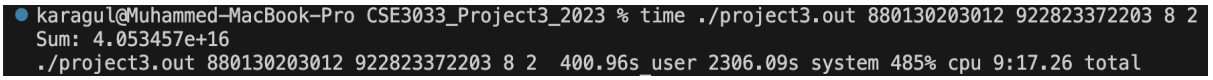
Method 1	Sum	User Time	System Time	Total Time	Cpu usage
c = 1	4.053457e+16	34.68 s	0.02 s	34.763 s	%99
c = 2	2.021011e+16	136.61 s	0.04 s	68.50 s	%199
c = 4	1.031340e+16	143.02 s	0.07 s	36.085 s	%396
c = 8	5.455896+e15	180.96 s	0.22 s	22.971 s	%788

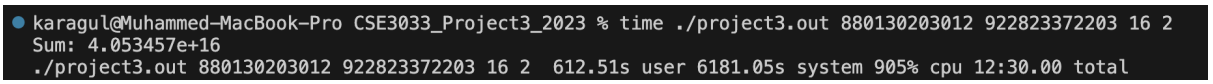
In Method 1, the results were different and inaccurate due to the lack of mutex. Also, due to the lack of mutex, not much time was spent in the kernel. Therefore the system time is very short. As expected, CPU usage and user time increased as the number of threads increased. When c = 1, the total time of the program took 34 seconds. When c = 8, it decreased to 23 seconds.

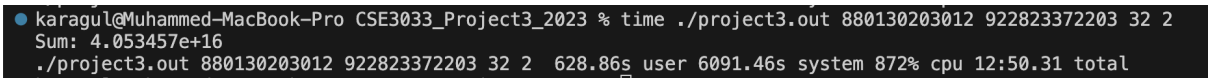
Method 2







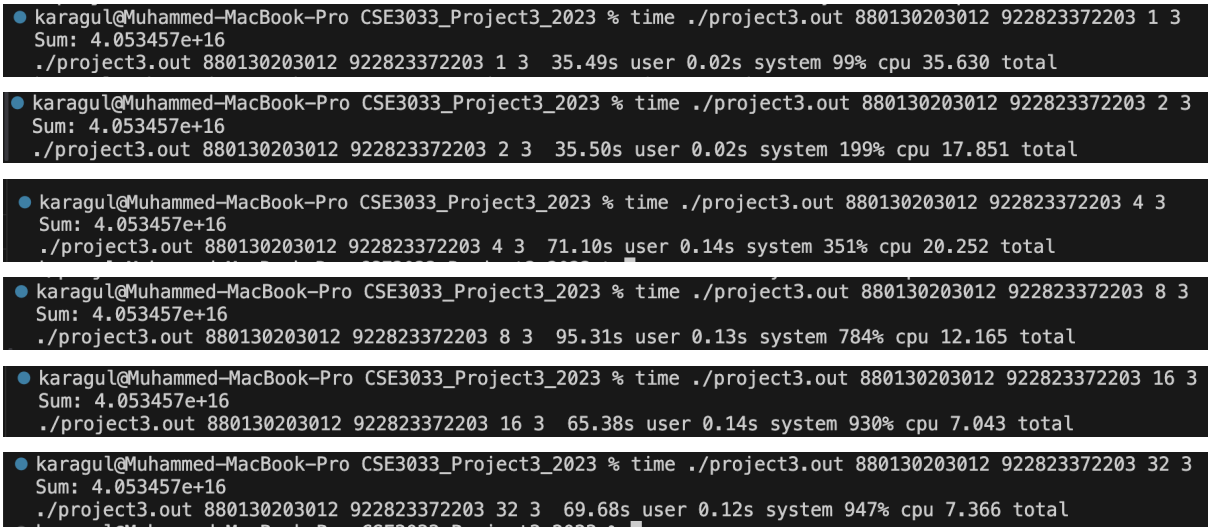




Method 2	Sum	User Time	System Time	Total Time	Cpu usage
c = 1	4.053457+e16	198.35 s	0.05 s	198.56 s	%99
c = 2	4.053457+e16	266.81 s	275.77 s	342.80 s	%158
c = 4	4.053457+e16	406.65 s	1005.97 s	475.62 s	%297
c = 8	4.053457+e16	400.96 s	2306.09 s	557.62 s	%485
c = 16	4.053457+e16	612.51 s	6181.05 s	750 s	%905
c = 32	4.053457+e16	628.86 s	6091.46 s	770.31 s	%872

In method 2, compared to method 1, we obtained accurate results in all inputs. We expected user time and system time to increase because we use mutex in this method. We did not expect the total time to increase, but as the number of threads increased, the program started to run slower. Total CPU usage would be very high with the increase in the number of threads as expected. We tried this method on a Linux computer with a not-very-powerful processor. The program ran for a very long time. That's why we took the data with Mac.

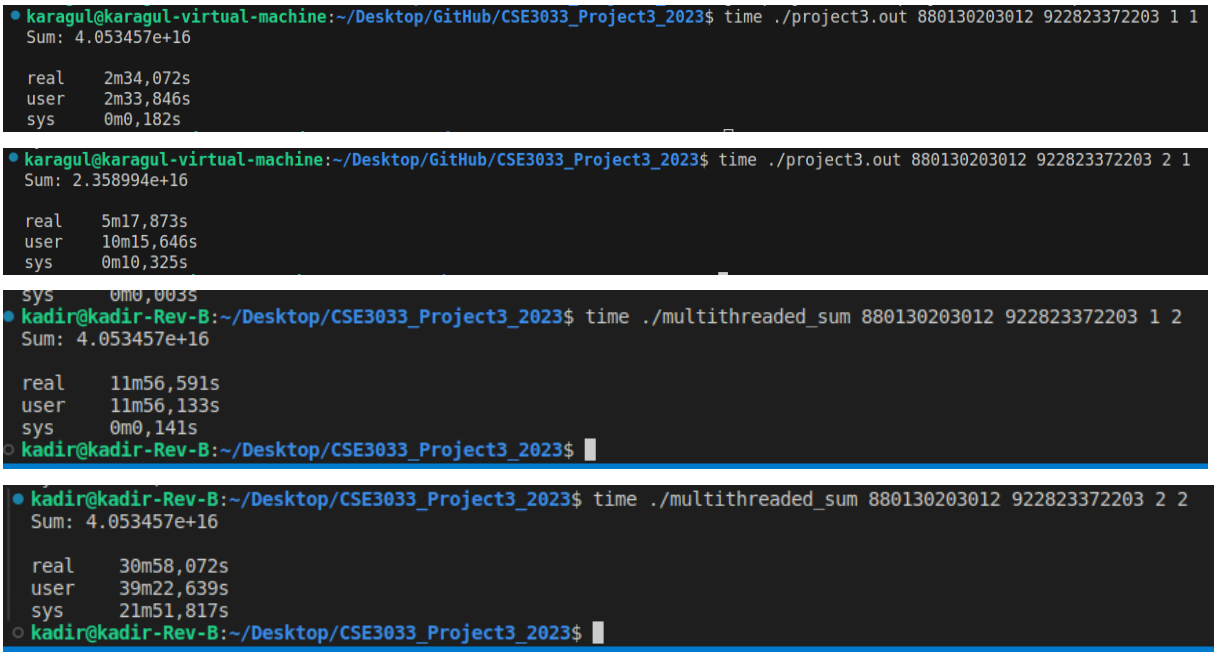
Method 3



Method 3	Sum	User Time	System Time	Total Time	Cpu usage
c = 1	4.053457+e16	35.49 s	0.02 s	35.63 s	%99
c = 2	4.053457+e16	35.50 s	0.02 s	17.851 s	%199
c = 4	4.053457+e16	71.10 s	0.14 s	20.252 s	%351
c = 8	4.053457+e16	95.31 s	0.13 s	12.165 s	%784
c = 16	4.053457+e16	65.38 s	0.14 s	7.043 s	%930
c = 32	4.053457+e16	69.68 s	0.12 s	7.366 s	%947

Method 3 was the fastest and most accurate method. It worked very fast compared to Method 2. It gave more accurate answers than Method 1. Therefore, method 3 was the best method among these 3 methods. As the number of threads increased, user time increased. This is a normal result. System time was quite short despite the increase in the number of threads. For this reason, the total time of the program was also quite short. We expected the program to speed up as the number of threads increased, and it did. The program that took 35 seconds when c = 1 took 7 seconds when c = 32. We learned how useful threads are.

Other Outputs



These are some outputs that ran in linux. The program took too long to run because the Linux computer had a bad processor.

To Compile Code:

- gcc project3.c -o project3.out -lm -pthread

To Run Code:

- time ./project3.out `<startOfTheRange>` `<endOfTheRange>` `<threadNumber>` `<methodNumber>`