CPSC457 - PRINCIPLES OF OPERATING SYSTEMS

University of Calgary
Assignment 3

Instructor: Pavol Federl
TA: Sina Keshvadi
Student: Omar Qureshi
ID: 10086638

Q1 - Suppose a program spends 2/3 of time doing I/O. Calculate the CPU utilization when running 5 copies of such a program at the same time. Assume one single-core CPU, and all I/O operations are executed in parallel.

Utilization = $1 - p^n = 1 - (0.6666)^5 = 0.868 = 86.8\%$, where p is the probability of the process waiting on I/O.

Q2 - see count.cpp

Q3 - Time the original program as well as your solution on three files from Appendix 2: medium.txt, hard.txt and hard2.txt. For each input file you will run your solution 6 times, using different number of threads: 1, 2, 3, 4, 8 and 16. You will record your results in 3 tables. The 'Observed timing' column will contain the raw timing results of your runs. The 'Observed speedup' column will be calculated as a ratio of your raw timing with respect to the timing of the original program. Once you have created the tables, explain the results you obtained. Are the timings what you expected them to be? If not, explain why they differ.

Test file: easy.txt			
#threads	Observed timing	Observed speedup compared to original	Expected speedup
Original program	0.004s	1.0	1.0
1	0.004s	1.0	1.0
2	0.003s	1.33	2.0
4	0.003s	1.33	4.0
8	0.004s	1.0	8.0
16	0.004s	1.0	16.0

Test file: medium.txt			
#threads	Observed timing	Observed speedup compared to original	Expected speedup
Original program	32.049s	1.0	1.0
1	33.072s	0.97	1.0
2	18.076s	1.77	2.0
4	11.936s	2.69	4.0
8	8.482s	3.78	8.0
16	7.199s	4.45	16.0

Test file: hard.txt			
#threads	Observed timing	Observed speedup compared to original	Expected speedup
Original program	10.764s	1.0	1.0
1	10.893s	1.0	1.0
2	11.193s	0.96	2.0
4	10.874	1.0	4.0
8	10.898s	1.0	8.0
16	11.160s	0.96	16.0

Test file: hard2.txt			
#threads	Observed timing	Observed speedup compared to original	Expected speedup
Original program	10.843s	1.0	1.0
1	11.078s	0.98	1.0
2	11.060s	0.98	2.0
4	11.064s	0.98	4.0
8	11.412s	0.95	8.0
16	11.122s	0.98	16.0

Explanation:

In my implementation, I try to choose the 80% route. I am assigning one thread to each individual number.

As such, in the easy.txt, there is not much gain to be had. This is because each individual number is small so the isPrime algorithm does not take very long. A single thread is able to quickly compute the numbers. In my program, with the additional overhead of the thread creation, it becomes even with the single thread program.

In the medium.txt, there is an improvement. This is because each individual thread can compute a large number on their own and this improves the concurrency of the isPrime function, whereas one thread would have to go execute that function at a time.

In the hard.txt and hard2.txt, we have one huge number and several small numbers. With the given implementation, this causes no speedup to be seen. This is because one thread bottlenecks on computing the large number, and the other threads finish and are then idle.