Processes v Threads

Ben Soer

A00843110

BTECH Set 6D

COMP 8005

After much experimentation, processes have come out to be slightly more effective then threads. In theory this appears to be a plausable outcome, but is not the one that was expected. Due to the lighter nature of threads in comparison to processes I hypothesised they would likely come out faster. But weight in this hypothesis was in terms of how long it would take to startup the process or thread. Concidering the architecture of most CPU's the results in hinesite appears more obvious then what I had predicted.

In thoery, it would make sense that processes would be more effective, given the timesliced system used by most CPU's. Using threads, the program only gets a single CPU timeslice to execute its threads, which attempt to work in parallel to complete the computations. Processes on the other hand each get a time slice and with the use of multiple cores can aswell work in parallel but over multiple timeslices. This gives the program conciderable more running time vs the idle time incurred by threads having to run within a single timeslice and then having to wait until the process had another turn.

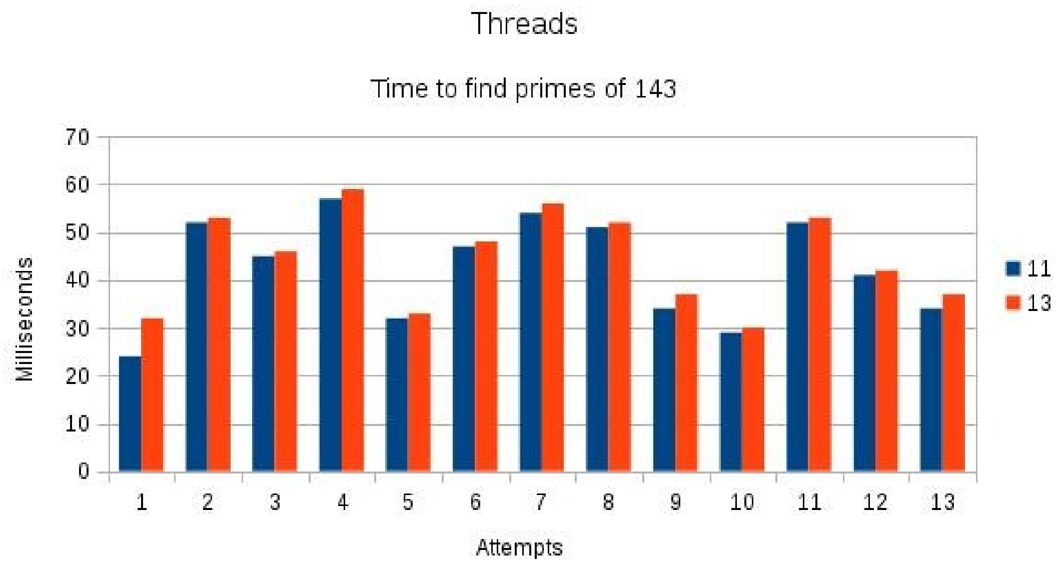
The calculations used to incure work was the decomposition of the product of two prime numbers. Instead of using loops during the calculation though, the work was split up so that it would create more individual tasks, and thus would test more the efficiency of the parallel functionality of processes and threads. This would reduce the mathematical intensity of a single processes task, but puts more reliance on the thread and process management in terms of speed of completion. Each “task” that a process or thread recieves is a number and then a divisor. The thread or process then calculates if the number is divisible by the divisor, if so then divides the number and produces more tasks for that resulted value so that they can be tested as to whether it can be divided further. If they are divisble and the divisor comes out as 1 and the original number is greater then 1 at this point, it can be safely stated that this number is a prime number.

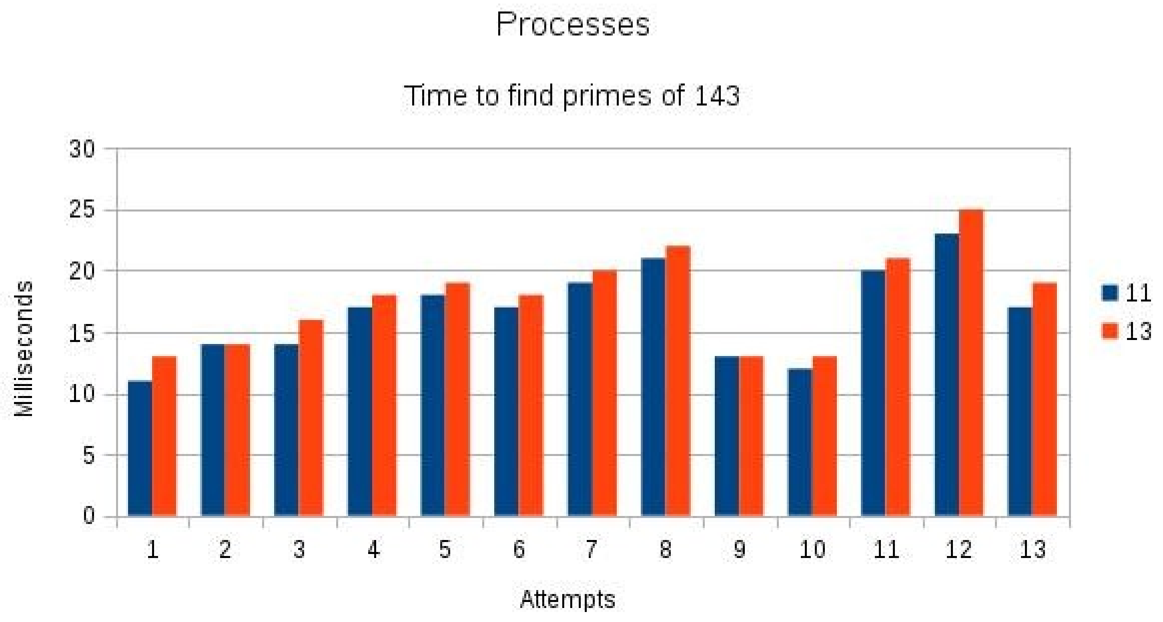
Note that the logic may appear to give an error in the event the user enters a product not made up of prime numbers. In this case the program will fail and will identify non-prime numbers as the roots of the product. This can then prove that the number given was not a product of primes. This would show though an incorrect use of the program and inaccurate benchmarking of the system, thus making it invalid and out of scope of the programs purpose.

I/O use was simply implemented by the use of aggressive logging. Both console logging and file logging is used on both programs to simulate high I/O. None of the console logging nor file writing is protected with mutexes or semaphores and thus leaves the fight over the I/O resources directly in the hands of the processes and threads. Although file writes are atomic, this puts extra strain specifically on the processes and thread parallism, testing their efficiency.

I/O to the file contains two unique measurements by the program for examination of efficiency. The first and most frequent numbers are the time it takes for a process or thread to complete an evaluation of the number and divisor. On average, both threads and processes in this area act relatively the same. The large difference comes from the second value which calculates the time it took to finaly find a root prime. This measurement is most valuable because it shows how effective the processes vs the threads were in taking as much of the CPU resources as possible, to come to the calculation.

Below diagrams the time it took to find the prime numbers 11 and 13 that made up the product of 143. The Y-Axis marks how many milliseconds it took to find the prime and the X-Axis marks each execution attempt of the program. Colored in blue and red are each prime number that make up the product. The first diagram is attempts using threading, the second uses processes.





From these charts we can see Threads average finding the prime numbers on average around 40ms. Process though, average finding the prime numbers in 15-20ms. The difference is quite substantial as the average of processess computation time is still better then the Thread's best attempt (attempts number 1) where it finds 11 in roughly 24 ms.

In summary, we can safely state that processes are more efficient and effective in their use of

CPU resources. Through the use of a process/thread straining, math and I/O intensive architecture there is a significant time difference between the two parallel computing tools.