erikvs Lab 7

The code produced for Lab 7 and the report has information taken from lectures and material given in lectures at Hiof in addition to Modern operating Systems written by Andrew S. Tanenbaum and Herbert Bos.

Exercise 1:

a)
Yes, but there is a point where p number of threads would have the opposite effect, decreasing efficiency.

b)
The definition of parallel programming is to split the work to utilize resources more efficiently.

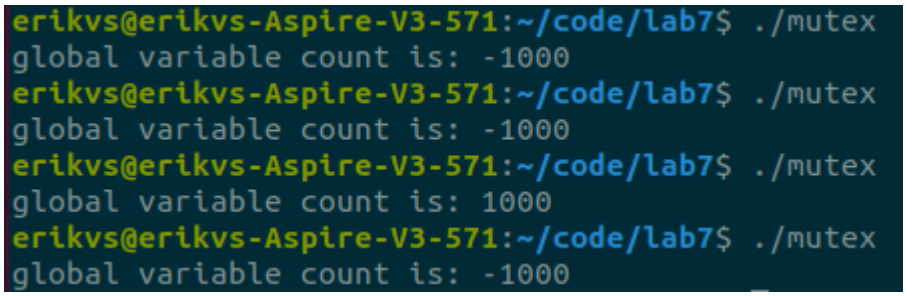
With each thread creation comes a small amount of overhead. Each thread has its own stack in the user space. Each thread has its own thread table containing the threads program counter, stack pointer, registers and state among other things. Swapping between threads is also something that requires resources, even if it is more efficient than kernel mode switching. When a thread has completed the work, it must be destroyed and this also requires resources. It is my understanding that the sum of this overhead is a very small amount of resources, but there is an amount to account for.

We are working under the assumption that the **threaded execution** is **equal to** the **serial execution** time **divided by** the **number of threads**. This is possible to achieve given that the number of threads created reflects the work needed to be done. It is not possible for me to work with exact numbers, since I cannot test the theory in practice by running the code, so I will use your example: If you have 24 hours of work needed to be done, it can be done faster with 2 people – but it will cost a bit more to hire another worker. The same amount of work can be done even faster with 10.000 workers, but the resources required to pay them would bankrupt you.

We are working with finite resources and a given amount of work to do. For a given amount of work in an X program, there comes a point where the number of threads adds enough overhead to the process to counter the efficiency gained from parallel programming.

Exercise 2:

a)



The program “initiates a race” between two threads to enter critical section and run their assigned functionality. Thread 1 runs the function Increase(), which increases count 10 from the base of 0 per iteration of the for loop to a total of 1000. Thread 2 runs the function Decrease() which decreases the count by 10 from the base of 0 per iteration of the for loop to a total of –1000. Whichever thread enters the critical section first will complete its for loop and then the threads are joined into main which prints the updated value of count.