# Multiprocessing

## Computer Architecture & Organisation- Assignment 1

Sachin Soman 19200494

# Introduction

Multitasking, multithreading, and multiprocessing. These three terms are often used in the overlap. I will first try to explain the three concepts before moving on to the assignment.

**Multi-tasking** is the process by which an operating system gives CPU time to each process on some form of schedule. Say we have a single-core CPU running two programs. In that case, the OS will give resources to program 1 for some time then pause the program and give the CPU resource to the next program. This process of switching happens so fast that it gives an illusion that programs are running simultaneously.

**Multithreading** is the process by which a single process can spawn multiple subprocesses that may run in parallel in a multi-core CPU or have threads that can be scheduled to get CPU resources by the OS in single-core processors. An advantage of a thread is that it shares the resources with other threads therefore there are fewer overheads compared to context switching between processes. An example of multiple threads can be different tabs in a browser. Where the browser is a single process and tabs are the threads of the process.

**Multi-processing** is a technique by which the process can leverage the multiple cores in a CPU to run different processes in parallel. This means we can split a task between different processors and execute the job in parallel making executions much faster.

An interesting thing that needs to be noted is that to utilize multi-threading or processing properly we need to design codes in a certain way otherwise can lead to unintentional results a classic example is the phenomenon of **a race condition** where two processes try to work on same variable resulting in unintended results. To avoid this we use the concept of semaphores and locks. Default python interpreter to avoid such problems do not allow multiple threads to execute simultaneously using Global Interpreter Lock (GIL). Python multiprocessing module, allows us to spawn multiple subprocesses to avoid some of the GIL disadvantages.

In this assignment, we will look at multi-processing how effective it is at speeding up tasks and some observations on its capabilities.

# Observations

Our first task in the assignment was to run a function each time with a different number of processes and to observe the speed up. I have already explained the code and specifics in a notebook, therefore, I will delve directly into my observations here.

- The first observation was that there was a significant increase in performance when we started using multi-processing. This is because we started using more cores of CPU with multiple processes. Parallel execution.

- The performance kept on increasing as we increased the number of processes and cores used.
- The increase in performance, however, started to plateau as we increased the number of processes beyond the number of available cores.
- Some examples the performance started to degrade as we increased the number of processes greater than the number of available cores
- The decrease in performance when the processes exceed the number of the core is due to the overheads generated by context switching that arises when processes have to share core via scheduling

The experiment was repeated for the same function with the different numbers of inputs and we observed the same trends. The next thing we did was try the experiment on a factorial generation function. This too yielded a similar result. Providing more evidence to our observations.

## Conclusion

We were able to see that multiprocessing significantly improved performances of the tasks we provided. The speedup was mainly due to the parallelization of processes and giving each process to different cores of the CPU. However, we see that this improvement is doesn't keep increasing as we increase the number of processes. We see a kind of plateau in gain. This should be expected as we cannot have more than 4 parallel processes running on a quad-core processor and any more processes mean we need to do CPU scheduling which creates more overhead. This is why in some cases of our experiment we see a decrease in performance when we raised the number of processes to six.

Another observation was that the greatest gain was when we used 3 processes even though the performance increased with 4 processes. This might be because the laptop has 4 cores and three cores can be dedicated to doing the process while the fourth processor may be used for system-level tasks. This is only a hypothesis and does not have sufficient evidence to back the claim.
Thus we can safely say that multi-processing improves performance through parallelization and utilization of available cores. However, as we increase the number of the process we also increase the overheads hence we need to find a proper trade-off point.
Also, not all programs can utilize multiprocessing as some programs must be run in series to achieve their results.

## Reference


- Python Multiprocessing Tutorial: Run Code in Parallel Using the Multiprocessing Module By Corey Shaffer
- Introduction to Parallel Processing Python