**Multiprocessing**

The “*Multiprocessing”* Python module provides functionality for distributing work between multiple processes, taking advantage of multiple CPU cores and larger amounts of available system memory.

It offers concurrency by sidestepping the Global Interpreter Lock (GIL) by using subprocesses instead of threads.This solves the issues in threading module.Each subprocess is an actual process and we can safely consider that it runs parallely.

Processes are heavier than threads and do not share anything.They communicate through IPC(Interprocess Communication Channel).Multiprocessing module , in particular, communicates using queues(Simple Queue and Joinablequeue) and pipes to communicate.However, though not advised,we can also have shared states.There are a couple of ways to do this, either by shared memory map or server process method.

Shared memory comes in the form of the Value and Array classes, and their names are indicative of what they're used for. Updates to a Value or Array object will be immediately visible to other processes with access to that object.Proper use of synchronization primitives is important when using shared memory.The server process method has a Manager object which controls the process and makes sure any change in shared object gets updated in all processes.Manager can maintain shared dicts,lists,namespaces etc.

Moreover,Multiprocessing has clones of most of the threading modules.For example:Event for signalling between processes,Lock to control access to resources, Condition to synchronize operations etc.Another important part of multiprocessing is pool.It has methods like .apply, apply\_async, .map etc.

**Advantages of multiprocessing-**

* Makes complete use of multiple CPU’s and cores:

--->GIL (Global Interpreter Lock) never allows more than one thread to run at a time.So,the resources were under utilised.Multiprocessing does not have any such Lock and so,it creates and runs multiple processes asynchronously and hence uses multi core processes.

* Avoid GIL limitations:

---> In the case of threading, even when one process gets corrupted the GIL got locked and disturbed the entire process.Multiprocessing can safely overcome this.

* Child processes are interruptible/killable unlike greenlets which are not preemptive.
* Faster than threading:

--->In threading though multiple threads are spawned , because of GIL only one thread actually runs at a time.As multiprocessing side stepped GIL,it can run processes parallely.However practically , multiprocessing is faster mostly in the cases of CPU Bound tasks only.

Points to consider:

* Though processes are heavier than threads i.e each process has it’s own address space etc,in cases where there is a lot destroying and creating of threads involved , processes can be preferred.
* Multiprocessing works only if the correct resources are provided i.e. multi processors.In systems where there is just one processor there is no point of implementation of multiprocessing.
* Gevent (uses greenlets) does not need multi core processors but works on the principle of co-operative multitasking.It may run in comparable time with regard to IO bound activities.So,in these cases it may be better to use packages other than multiprocessing.

Conclusion : It is safe to assume that multiprocessing is better in CPU bound tasks and gevent while performing IO bound tasks.

The following code examples and their run times prove the above statements.

Task:Finding the prime factors of 1,00,000 numbers

* 1 process : 0.796 seconds
* 2 processes : 0.484 seconds
* 4 processes: 0.280 seconds
* 8 processes : 0.220 seconds

The above being CPU bound is improved by multiple processes.

On the other hand, a task of reading 40 urls takes 6.6 ms