Write-up on the global interpreter lock

From python.org the description of the global interpreter lock is the following:

> In CPython, the **global interpreter lock**, or **GIL**, is a mutex that prevents multiple native threads from executing Python bytecodes at once. This lock is necessary mainly because CPython's memory management is not thread-safe. (However, since the GIL exists, other features have grown to depend on the guarantees that it enforces.)

This in turn has prevented the clients to be able to run in parallel when running within the same python environment. Namely, when you initialize a python program running something such as “python myprogram.py” you are forking a new osthread to run the python environment, and then in turn, when you execute a subroutine within the python program (particularly one whose purpose satisfies the need of the entire program), it’s okay that you are blocking other new processes from executing simply because all the work needed to get done can be done within the single threaded nature as enforced by the GIL. This right here was the relevant information for why you cannot spin up multiple http server clients from a single python parent thread, as since only a single thread can run at a time, if work is being processed in anyway for a single one of the http threads, requests to the other servers could not possibly be picked up.

However, one should not that you can multithread, but not parallelize http requests to a single server by issuing a wait on incoming requests. Additionally though, due to the GIL this is very inefficient as you are constantly switching between threads and in turn multithreaded programs in python (especially in certain versions of python 2.7) are substantially slower than single threaded programs, or ones where you have separate processes for queuing and data crunching (especially when both can be parallelized in a distributed fashion across multiple main python threads and machines).