Concurrency with Processes, Threads, and Coroutines

Python includes sophisticated tools for managing concurrent operations using processes and threads. Even many relatively simple programs can be made to run faster by applying techniques for running parts of the job concurrently using these modules.

`subprocess` provides an API for creating and communicating with secondary processes. It is especially good for running programs that produce or consume text, since the API supports passing data back and forth through the standard input and output channels of the new process.

The `signal` module exposes the Unix signal mechanism for sending events to other processes. The signals are processed asynchronously, usually by interrupting what the program is doing at the time the signal arrives. Signalling is useful as a coarse messaging system, but other inter-process communication techniques are more reliable and can deliver more complicated messages.

`threading` includes a high-level, object oriented, API for working with concurrency from Python. `Thread` objects run concurrently within the same process and share memory. Using threads is an easy way to scale for tasks that are more I/O bound than CPU bound.

The `multiprocessing` module mirrors `threading`, except that instead of a `Thread` class it provides a `Process`. Each `Process` is a true system process without shared memory, but `multiprocessing` provides features for sharing data and passing messages between them so that in many cases converting from threads to processes is as simple as changing a few `import` statements.

`asyncio` provides a framework for concurrency and asynchronous I/O management using either a class-based protocol system or coroutines. `asyncio` replaces the old `asyncore` and `asynchat` modules, which are still available but deprecated.

`concurrent.futures` provides implementation of thread and process-based executors for managing resources pools for running concurrent tasks.

hmac — Cryptographic Message Signing and Verification subprocess — Spawning Additional Processes

This page was last updated 2016-12-18.

Navigation

hmac — Cryptographic Message Signing and Verification
subprocess — Spawning Additional Processes

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?