multiprocessing — Manage Processes Like Threads

Purpose: Provides an API for managing processes.

The `multiprocessing` module includes an API for dividing work up between multiple processes based on the API for `threading`. In some cases `multiprocessing` is a drop-in replacement, and can be used instead of `threading` to take advantage of multiple CPU cores to avoid computational bottlenecks associated with Python's global interpreter lock.

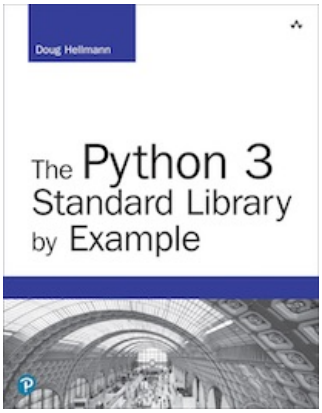
Due to the similarity, the first few examples here are modified from the `threading` examples. Features provided by `multiprocessing` but not available in `threading` are covered later.

- multiprocessing Basics
- Importable Target Functions
- Determining the Current Process
- Daemon Processes
- Waiting for Processes
- Terminating Processes
- Process Exit Status
- Logging
- Subclassing Process
- Passing Messages to Processes
- Signaling between Processes
- Controlling Access to Resources
- Synchronizing Operations
- Controlling Concurrent Access to Resources
- Managing Shared State
- Shared Namespaces
- Process Pools
- Implementing MapReduce

See also

- Standard library documentation for multiprocessing
- `threading` – High-level API for working with threads.
- MapReduce - Wikipedia – Overview of MapReduce on Wikipedia.
- MapReduce: Simplified Data Processing on Large Clusters – Google Labs presentation and paper on MapReduce.
- `operator` – Operator tools such as `itemgetter`.

This page was last updated 2016-12-17.



Get the book

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?