(/learn)

Introduction

This lesson lays the groundwork for discussing the various APIs and capabilities of the multiprocessing module.

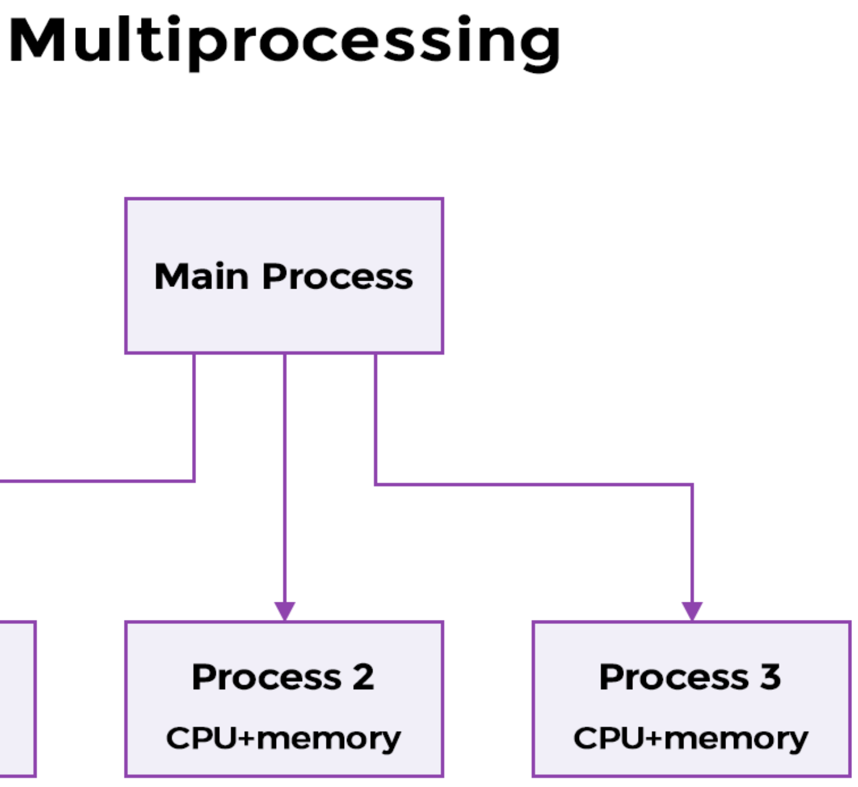
Introduction

Python offers the ability to execute tasks as processes using the **`multiprocessing`** module. Most of the APIs in the module mirror the APIs found in the **`threading`** module. In fact if you have a program written using the **`threading`** module APIs, it is trivial to change the program to work with the **`multiprocessing`** module.

The limitations of the global interpreter lock are to an extent addressed by the **`multiprocessing`** module. One of the most common critiques of Python has been its inability to schedule threads on multiple processors in a multicore system. So much so, that alternative implementations such as Jython, IronPython, etc have cropped up to address this shortcoming of the standard Python implementation. The **`multiprocessing`** module offers hope by allowing a program to spin-off tasks as separate processes that can then run on individual processors. This allows Python to be both concurrent and parallel, whereas with the threading module, Python is only concurrent and not parallel. Languages without a GIL in their design, such as Java, can exhibit both concurrency and parallelism using only threads on a multicore system.

Creating, managing and tearing down processes is more expensive than doing the same for threads. Furthermore, inter-process communication is relatively slower than inter-thread communication. Both these drawbacks may not make Python a practical technology choice for super-critical or time-sensitive use-cases.

Given the above context, let's explore the **`multiprocessing`** module and the APIs it offers.



Back
Quiz 2

Next
Process

✓ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/introduction__multiprocessing__python-concurrency-for-senior-engineering-interviews)