Short Python Processes Explanation

So what are processes and why do we care?

1. What Are Processes

A process refers to a computer program.

Every Python program is a process and has one default thread called the main thread used to execute your program instructions. Each process is, in fact, one instance of the Python interpreter that executes Python instructions (Python byte-code), which is a slightly lower level than the code you type into your Python program.

Sometimes we may need to create new processes to run additional tasks concurrently.

Python provides real system-level processes via the **multiprocessing.Process** class in the **multiprocessing** module.

The underlying operating system controls how new processes are created. On some systems, that may require spawning a new process, and on others, it may require that the process is forked. The operating-specific method used for creating new processes in Python is not something we need to worry about as it is managed by your installed Python interpreter.

The code in new child processes may or may not be executed in parallel (at the same time), even though the threads are executed concurrently.

There are a number of reasons for this, such as the underlying hardware may or may not support parallel execution (e.g. one vs multiple CPU cores).

This highlights the distinction between code that can run out of order (concurrent) from the capability to execute simultaneously (parallel).

- **Concurrent**: Code that can be executed out of order.
- **Parallel**: Capability to execute code simultaneously.

Processes and threads are different.

Next, let's consider the important differences between processes and threads.

2. Thread vs Process

A process refers to a computer program.

Each process is in fact one instance of the Python interpreter that executes Python instructions (Python byte-code), which is a slightly lower level than the code you type into your Python program.

> *Process: The operating system's spawned and controlled entity that encapsulates an executing application. A process has two main functions. The first is to act as the resource holder for the application, and the second is to execute the instructions of the application.*
> **— PAGE 271, THE ART OF CONCURRENCY, 2009.**

The underlying operating system controls how new processes are created. On some systems, that may require spawning a new process, and on others, it may require that the process is forked. The operating-specific method used for creating new processes in Python is not something we need to worry about as it is managed by your installed Python interpreter.

A thread always exists within a process and represents the manner in which instructions or code is executed.

A process will have at least one thread, called the main thread. Any additional threads that we create within the process will belong to that process.

The Python process will terminate once all (non background threads) are terminated.

- **Process**: An instance of the Python interpreter has at least one thread called the *MainThread*.
- **Thread**: A thread of execution within a Python process, such as the *MainThread* or a new thread.

You can learn more about the differences between processes and threads in the tutorial:

- Thread vs Process in Python

Next, let's take a look at processes in Python.

Life-Cycle of a Process

A process in Python is represented as an instance of the **multiprocessing.Process** class.

Once a process is started, the Python runtime will interface with the underlying operating system and request that a new native process be created. The **multiprocessing.Process** instance then provides a Python-based reference to this underlying native process.

Each process follows the same life cycle. Understanding the stages of this life-cycle can help when getting started with concurrent programming in Python.

For example:

- The difference between creating and starting a process.
- The difference between run and start.
- The difference between blocked and terminated

And so on.

A Python process may progress through three steps of its life cycle: a new process, a running process, and a terminated process.

While running, the process may be executing code or may be blocked, waiting on something such as another process or an external resource. Although not all processes may block, it is optional based on the specific use case for the new process.

1. New Child Process.
2. Running Process.
    1. Blocked Process (optional).
3. Terminated Process.

A new process is a process that has been constructed and configured by creating an instance of the **multiprocessing.Process** class.

A new process can transition to a running process by calling the **start()** method. This also creates and starts the main thread for the process that actually executes code in the process.

A running process may block in many ways if its main thread is blocked, such as reading or writing from a file or a socket or by waiting on a concurrency primitive such as a semaphore or a lock. After blocking, the process will run again.

Finally, a process may terminate once it has finished executing its code or by raising an error or exception.

A process cannot terminate until:

- All non-daemon threads have terminated, including the main thread.
- All non-daemon child processes have terminated, including the main process.

You can learn more about the life-cycle of processes in the tutorial:

- Process Life-Cycle in Python

Next, let's take a closer look at the difference between parent and child processes.

Child vs Parent Process

Parent processes have one or more child processes, whereas child processes is created by a parent process.

Parent Process

A parent process is a process that is capable of starting child processes.

Typically, we would not refer to a process as a parent process until it has created one or more child processes. This is the commonly accepted definition.

- **Parent Process**: Has one or more child processes. May have a parent process, e.g. may also be a child process.

Recall, a process is an instance of a computer program. In Python, a process is an instance of the Python interpreter that executes Python code.

In Python, the first process created when we run our program is called the 'MainProcess'. It is also a parent process and may be called the main parent process.

The main process will create the first child process or processes.

A child process may also become a parent process if it in turn creates and starts a child process.

Child Process

A child process is a process that was created by another process.

A child process may also be called a subprocess, as it was created by another process rather than by the operating system directly. That being said, the creation and management of all processes is handled by the underlying operating system.

- **Child Process**: Has a parent process. May have its own child processes, e.g. may also be a parent.

The process that creates a child process is called a parent process. A child process only ever has one parent process.

There are three main techniques used to create a child process, referred to as process start methods.

They are: fork, spawn, and fork server.

Depending on the technique used to start the child, the child process may or may not inherit properties of the parent process. For example, a forked process may inherit a copy of the global variables from the parent process.

A child process may become orphaned if the parent process that created it is terminated.

You can learn more about the differences between child and parent processes in the tutorial:

- Parent Process vs Child Process in Python

Now that we are familiar with child and parent processes, let's look at how to run code in new processes.