Daemon Processes Short explanation

1. Introduction

Every Python program is executed in a Process, which is a new instance of the Python interpreter.
This process has the name MainProcess and has one thread used to execute the program instructions
called the MainThread. Both processes and threads are created and managed by the underlying operating system.

Sometimes we may need to create new child processes in our program in order to execute code concurrently.

Python provides the ability to create and manage new processes via the multiprocessing.Process class.

In concurrent programming, we may need to execute sporadic, periodic or long-running tasks in the background.

A special type of process is used for background tasks, called a daemon process.

2. What is a Daemon Process

A daemon thread or a daemon process runs in the background.

Daemon is pronounced “*dee-mon*“, like the alternate spelling “*demon*“.

The idea is that background processes are like “daemons” or spirits (from the ancient Greek) that do tasks for you in the background. You might also refer to daemon processes as daemonic.

A process may be configured to be a daemon or not, and most processes in concurrent programming, including the main parent process, are non-daemon processes (not background processes) by default.

The property of being a daemon process or not may be supported by the underlying operating system that actually creates and manages the execution of threads and processes.

Daemon processes are helpful for executing tasks in the background to support the non-daemon processes in an application.

Uses of daemon processes might include:

- Background logging to file or database.
- Background data retrieval, updates, refresh.
- Background data storage to disk or database.

Background tasks can be varied in type and specific to your application.

Some properties of these tasks might include:

- **Sporadic**: Tasks that run only when specific conditions arise (e.g. ad hoc logging).
- **Periodic**: Tasks that run after a consistent interval (e.g. data save/load every minute).
- **Long-Running**: Tasks that run for the duration of the program (e.g. monitoring a resource).

Typically daemon processes execute non-critical tasks that although may be useful to the application are not critical if they fail (perhaps silently) or are canceled mid-operation.

By definition, a daemon process will not have control over when it is terminated.

The main parent process will terminate once all non-daemon processes finish, even if there are daemon processes still running. Therefore, the code it executes must be robust to arbitrary termination, such as the flushing and closing of external resources like streams and files that may not be closed correctly.

> *When a process exits, it attempts to terminate all of its daemonic child processes.*
>
> **MULTIPROCESSING — PROCESS-BASED PARALLELISM**

Note, a daemon process is an extension of a daemon thread to processes. You can learn more about daemon threads in the tutorial: