Introduction to Multithreading and Multiprocessing in Python

Pham Thi Ngoc Mai

Onnet - AHT

September 30th, 2021

Thread & Process

What is Process?

Process

- **Process** is a program in execution state
- Process Control Block (PCB) is the brain of process

Process Components

What is Thread?

Thread

- a single flow of execution
- belongs to a process
- can be considered as a lightweight process

Single-threaded process

- Default
- Only one thread per process

Multi-threaded process

- More than one thread per process
- Share memory allocation (heap, global data) among threads
- Different stack

Multi-threaded process

Thread vs Process

Process	Thread
heavy weight	light weight
context switches are time consumming	context switches are less time consumming
independent	interdependent
inter-communication is not easy	inter-communication is simpler

CPU-bound and I/O-bound processes

- A program is **CPU bound** if it would go faster if the CPU were faster
  - do mathematical computations
- A program is **I/O bound** if it would go faster if the **I/O subsystem** (disk, networking) was faster
  - waiting for Input/Output which can come from a user, file, database, network, etc.

Multithreading and Multiprocessing

Multithreading

Pros

- make responsive UI
- ideal option for I/O bound applications

Cons

- synchronization
- race condition & deadlock
- @___@ code

Multiprocessing

Pros

- get more work done in shorter period
- straight forward code
- take benefit form multiple CPUs & cores

Cons

- IPC is complicated and overhead
- large memory footprint

TLDR

You can use **threading** if your program is **network bound** or **multiprocessing** if it's **CPU bound**

Python and GIL

Back to the old days

- Python wasn't designed considering that personal computers might have more than one core.
- Python is OLDDDDDDDDDDD...
- World is one core :P, threading existed but not for computing power

→ Multi-threaded applications' problems :/

The **Infamous** feature of Python - GIL

What is GIL?

Global Interpreter Lock

- One massive log, on everything
- Only interpret one thread of Python at a time

→ Threads never run at the same time

→ silly, uneeded, behind the times, ruins things... (/pats)

Why **still** Python GIL? - The trade-off

Multi-core is **everywhere** now

Pros

- Readability First
- Reference Count as memory management
- Easy to get right
- No deadlocks
- I/O bound: ok
- CPU bound: single threaded
  the design decision of the GIL is one of the things that made Python as popular as it is today. - Larry Hasting

Why **still** Python GIL? - The trade-off

It isn't Easy to Remove the GIL - Guido van Rossum

In Odoo