Multi-Tasking -- Multi-threading in Python

A deep dive into multi-threading and multi-processing with Python and how they are related to concurrency and parallelism.

Here, we will discuss :

- **Difference between concurrency and parallelism**
- **Process vs. Thread**
- **Multi-Threading in Python**

What is Multitasking in Python ?

Let's imagine we got slow python code

- **In such situations we try to optimize the code itself. Things such as using proper algorithms and data structures.**
- **If it's still not enough we can optimize our code to use more of the hardware**

In layman's terms, it's multitasking

- Multitasking is the ability of an operating system to perform multiple tasks simultaneously

- For example, you are working on a Jupyter Notebook, downloading a movie, and listening to a song

- All these tasks are performed by the same OS and in sync

- There are two types of multitasking :

  `1.Process-based - Parallelism`

  `2.Thread-based - Concurrency`

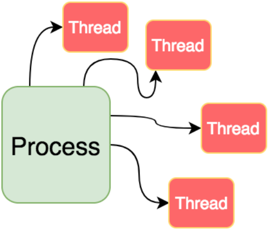
Terminologies :

Before we dive in let's understand what these terms mean :

- **Program :** A program is an executable file which consists of a set of instructions toperform some task and is usually stored on the disk of your computer.

- **Process :** A process is what we call a program that has been loaded into memory along with all the resources it needs to operate. It has its own memory space.

- **Thread :** A thread is the unit of execution within a process. A process can have multiple threads running where each thread uses the process's memory space and shares it with other threads.



- **Core :** The CPU's processor. This term refers to the hardware component of your CPU

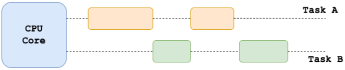
A core can work on a single task; multi-core processors can perform multiple tasks at once.

Concurrency :

- Concurrency is fake multitasking, meaning that you don't run things simultaneously.

- You instead take turns and hence making it look like you're running things simultaneously.

- **Concurrency is when tasks start, run and complete in overlapping periods,on a single-core server.**



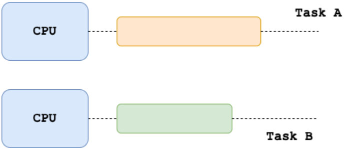
- **Concurrency in python can be achieved through**

  `1.Multi Thereading`

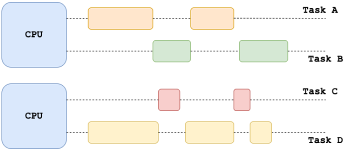
  `2.Asyncio (Asynchronous Programming)`

Parallelism :

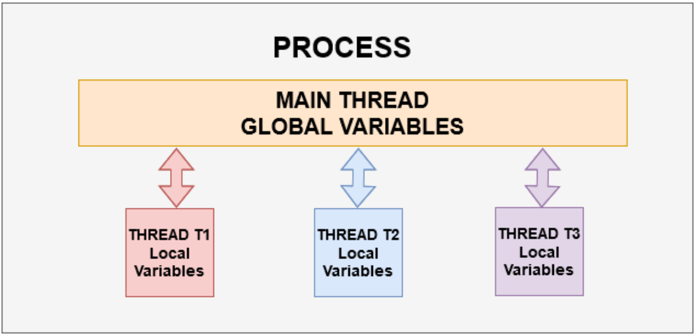
- **Parallelism is true multitasking, meaning that tasks literally start and run at the same time.**
- **This is done using multiprocessing, where you use multiple CPU cores to distribute tasks accordingly**
- **This doesn't "break" up the code into parts, each core has a complete running copy of your program.**
- **In multi-core environments, each core can execute one task at exactly the same time**
- **With parallelism, we are able to maximise the use of hardware resources**



Additionally,both concurrency and parallelism could be combined during task execution



Multithreading in Python



- **Multithreading is defined as the ability of a processor to execute multiple threads concurrently,**

- **All threads of a process share global variables and the program code and shares the same CPU and memory**

- **It is achieved using frequent switching between threads. This is termed as context switching.**

- **In context switching, the state of a thread is saved and state of another thread is loaded**

whenever any interrupt (due to I/O or manually set) takes place

- **However, because of the GIL in Python, not all tasks can be executed faster by using multithreading**

- **A simple example of multi-threading is downloading multiple files from the Internet.**

- **There are two main modules which can be used to handle threads in Python: :**

  1. `The thread module, and`

  2. `The threading module`

The Thread Module

- **The syntax to create a new thread using this module is as follows :**

```
thread.start_new_thread(function_name, arguments)
```

- **However, the thread module has long been deprecated. Starting with Python 3**

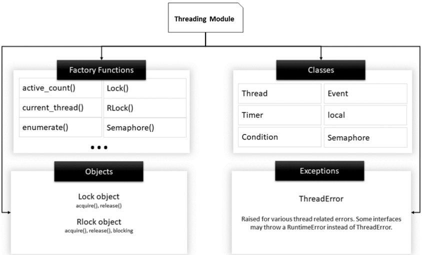
- **It has been designated as obsolete and is only accessible as _thread for backward compatibility.**

- **We should use the higher-level threading module for applications which you intend to deploy**

The Threading Module :

- **It is the high-level implementation of threading in python and used for managing multithreaded applications**

- **It provides a wide range of features when compared to the thread module.**



Lets Code:

In [17]:

```
#This program includes two separate tasks (functions) as follows
from time import sleep, perf_counter
def calc_square(numbers):
    for n in numbers:
        print(f'\nCaluclating Square : {n} ^ 2 = {n*n}')
        time.sleep(1)

def calc_cube(numbers):
    for n in numbers:
        print(f'\nCaluclating Cube : {n} ^ 3 = {n*n*n}')
        time.sleep(1)
```

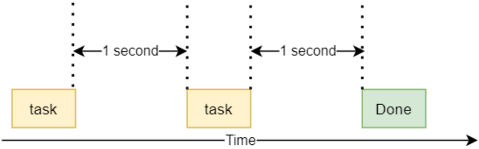
Single-threaded applications

In [18]:

```
#Single-threaded applications
numbers = [2, 3, 5, 8]

start = perf_counter()

calc_square(numbers)
calc_cube(numbers)

end = perf_counter()

print(f'\nIt took {end-start: 0.2f} second(s) to complete.')
```

```
Caluclating Square : 2 ^ 2 = 4

Caluclating Square : 3 ^ 2 = 9

Caluclating Square : 5 ^ 2 = 25

Caluclating Square : 8 ^ 2 = 64

Caluclating Cube : 2 ^ 3 = 8

Caluclating Cube : 3 ^ 3 = 27

Caluclating Cube : 5 ^ 3 = 125

Caluclating Cube : 8 ^ 3 = 512

It took  8.09 second(s) to complete.
```

- **The following diagram illustrates how the program works :**



- **First, the calc_square( ) function executes and sleeps for one second. Then it executes**

calc_cube( ) and also sleeps for another second. Finally, the program completes.

- **When the function calls the sleep() function, the CPU is idle.**

- **In other words, the CPU doesn’t do anything, which is not efficient in terms of resource utilization.**

- **This program has one process with a single thread, which is called the main thread**

- **Because the program has only one thread, it’s called the single-threaded program.**

- **Remember, every process has one “main thread” always running.**

Multi-Threaded Program :

- **A threading module is made up of a Thread class, which is instantiated to create a Python thread.**

- **The Thread ( ) accepts many parameters. The main ones are :**

target: specifies a function (fn) to run in the new thread.

args: specifies the arguments of the function (fn). The args argument is a tuple.</pre>

- We start the thread by calling the start () method of the Thread instance

- By calling the join( ) method, the main thread will wait for the second thread to complete before it is terminated.

- The main thread creates the child thread objects and also initiates it.

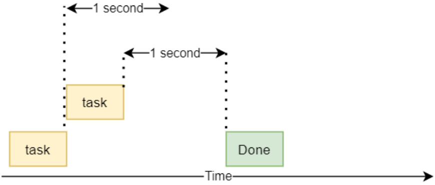
In [19]:

```
import threading

start = time.time()

#Instantiating Thread Class
square_thread = threading.Thread(target=calc_square, args=(numbers,))
cube_thread = threading.Thread(target=calc_cube, args=(numbers,))

#Starting the Thread
square_thread.start()
cube_thread.start()

#Joining the threads
square_thread.join()
cube_thread.join()

end = time.time()

print(f'\nIt took {end-start: 0.2f} second(s) to complete.')
```

```
Caluclating Square : 2 ^ 2 = 4

Caluclating Cube : 2 ^ 3 = 8

Caluclating Square : 3 ^ 2 = 9

Caluclating Cube : 3 ^ 3 = 27

Caluclating Square : 5 ^ 2 = 25

Caluclating Cube : 5 ^ 3 = 125
```

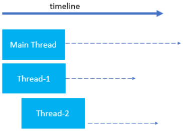
```
Caluclating Square : 8 ^ 2 = 64

Caluclating Cube : 8 ^ 3 = 512

It took  4.07 second(s) to complete.
```

The following diagram shows how threads execute:

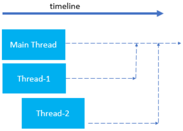


Note : Join ( ) should be placed in the main thread. Join blocks the calling or the main thread until

the execution of the thread that is joined also terminates.



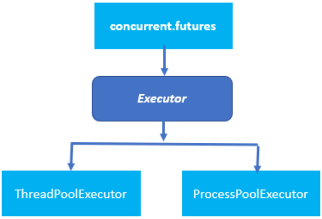
Without Join



With Join

ThreadPool Executor

- **ThreadPoolExector is an easy way to implement and spawn multiple threads using concurrent.futures.**



- **concurrent.futures has an abstract class Executor, and it has two concrete subclasses**

```
ThreadPoolExecutor: For multi Threading


ProcessPoolExecutor: For multi Processing</pre>
```

- `ThreadPoolExecutor class exposes three methods to execute threads asynchronously :`

`The submit() method takes a function and executes it asynchronously.`

`map() - execute a function asynchronously for each element in an iterable.`

`shutdown() - shut down the executor.`

In [24]:

```python
import requests
import time
import concurrent.futures

img_urls = ["https://unsplash.com/photos/agzJY5jrsAw/download?force=True",
            "https://unsplash.com/photos/4rDCa5hBlCs/download?force=True",
            "https://unsplash.com/photos/jFCViYFYcus/download?force=True",
            "https://unsplash.com/photos/Y8lCoTRgHPE/download?force=True",
            "https://unsplash.com/photos/4KrQq8Z6Y5c/download?force=True"]

t1 = time.perf_counter()
def download_image(img_url):
    img_bytes = requests.get(img_url).content
    print("Downloading..")

# Download images 1 by 1 => slow
for img in img_urls:
    download_image(img)
t2 = time.perf_counter()
print(f'Single Threaded Code Took :{t2 - t1} seconds')

print('*'*50)

t1 = time.perf_counter()
def download_image(img_url):
    img_bytes = requests.get(img_url).content
    print("Downloading..")

# Fetching images concurrently thus speeds up the download.
with concurrent.futures.ThreadPoolExecutor(3) as executor:
    executor.map(download_image, img_urls)

t2 = time.perf_counter()
print(f'MultiThreaded Code Took:{t2 - t1} seconds')
```

```
Downloading..
Downloading..
Downloading..
Downloading..
Downloading..
Single Threaded Code Took :8.645940800000972 seconds
**************************************************
Downloading..
Downloading..
Downloading..
Downloading..
Downloading..
MultiThreaded Code Took:3.4672711999992316 seconds
```

© Nitheesh Reddy