

Multithreading

The Unknown Truth of Python (2.7)



- Ritu Chawla Mehra
Xoriant Solutions, Mumbai
30-11-2017

What will you learn in this session

**Concurrency &
Multithreading**

**GIL
Global
Interpreter Lock**

GIL MYSTERY

**The Case Study
Where to use & where not to use Multithreading**

**Multithreading
Examples**

**Multiprocessing
Example**

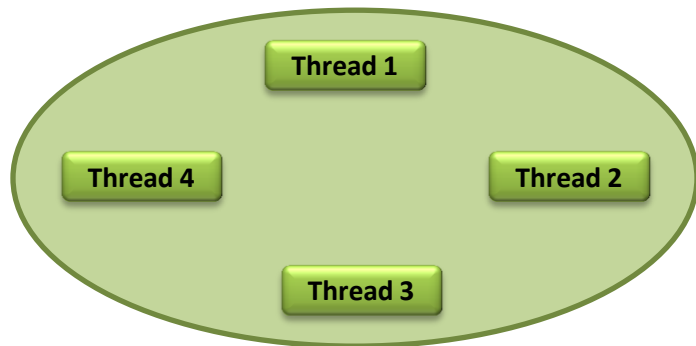
**Multiprocessing
CPU Bound Task**

Concurrency & Multithreading



Concurrency:

- Two or more tasks **start and complete** in overlapping time period.
- **Manage** access to shared state(code) from different tasks.



Multithreading:

- Process of **executing multiple threads concurrently**.
- Shares same piece of code, data & resources (memory).
- Maintain their own stack.

Global Interpreter Lock: Mystery behind it



GIL:

- Mutex lock which prevents multiple threads from executing Python code at once.
- Effectively it allows only one thread to execute at a given point of time.

The Purpose :

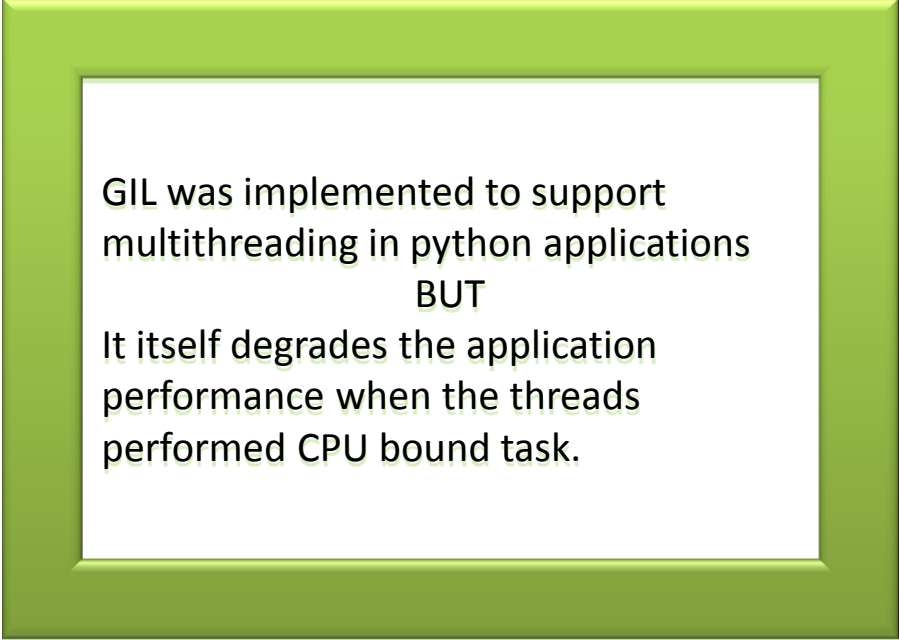
- To support multithreaded Python programs as “**The Python Interpreter**” is not fully thread-safe.

How does it work:

- The GIL must be held by the current thread before it can safely access Python objects.
- Python's standard library releases the GIL whenever there is any blocking i/o operation (network i/o, reading, writing) , so that other threads can execute in the mean time .
- When it's not an I/O operation, say some CPU bound task of computations, it won't get released and other threads keep on waiting.



MYSTERY



GIL was implemented to support
multithreading in python applications
BUT
It itself degrades the application
performance when the threads
performed CPU bound task.



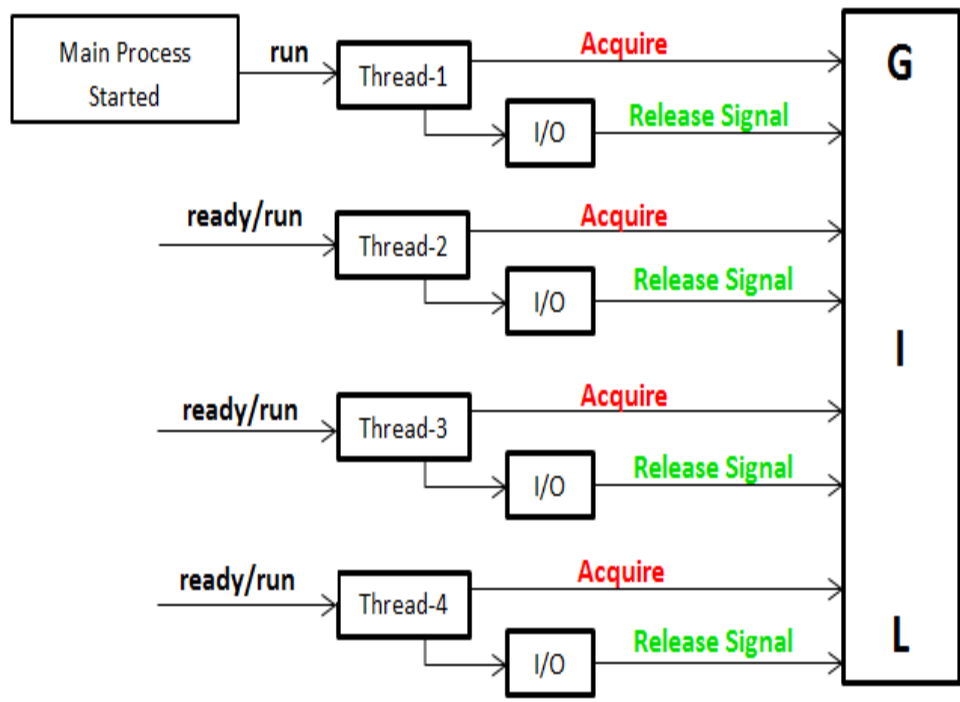
**Should we use
Multithreading ?**

R
E
S
P
O
N
S
E





When to “use/not-use”
multithreading and “Why”



YES

- These tasks don't occupy CPU when they are performing operations.
- Examples : Network requests, Downloading , Copying, Moving (data/files/etc).
- GIL release signals are handled by thread library and operating system



Multithreading Examples

```
def access_url_data(curr_url):  
    print("Start: {}".format(threading.current_thread().name))  
    res = requests.get(curr_url)  
    print("End : {}".format(threading.current_thread().name))
```

```
def worker_thread_queue():  
    while True:  
        current_url = q_url.get()  
        access_url_data(current_url)  
        q_url.task_done()
```

```
q_url = Queue.Queue()  
for i in range(thread_count):  
    obj_thread = threading.Thread(target=worker_thread_queue)  
    obj_thread.daemon = True; obj_thread.start()
```

```
start_time = time.time()  
url_list = ["https://www.python.org", "https://www.python.org",  
            "https://www.python.org", "https://www.python.org"]
```

```
for current_url in url_list:  
    q_url.put(current_url)  
q_url.join()
```

```
print "\nthread_count: " , thread_count  
print "Start Time (sec): " , start_time  
print "End Time (sec): " , time.time()
```

```
m,s = divmod(time.time() - start_time,60)  
print "Time taken to execute: " , s
```

```
$ python I_O_Bound_MultiThreading.py  
Start: Thread-1  
End : Thread-1  
Start: Thread-1  
End : Thread-1  
Start: Thread-1  
End : Thread-1  
Start: Thread-1  
End : Thread-1  
Start: Thread-1  
End : Thread-1  
  
thread_count: 1  
Start Time (sec): 1511554684.93  
End Time (sec): 1511554689.28  
Time taken to execute: 4.35000014305
```

```
$ python I_O_Bound_MultiThreading.py  
Start: Thread-1  
Start: Thread-2  
Start: Thread-3  
Start: Thread-4  
End : Thread-3  
End : Thread-2  
End : Thread-1  
End : Thread-4  
  
thread_count: 5  
Start Time (sec): 1511554704.93  
End Time (sec): 1511554706.76  
Time taken to execute: 1.83000016212
```

CPU Bound Task - Multithreading

```
def calculate_sum(num):
    sum_total = 0
    init = 2
    while init <= num:
        sum_total += init
        init += 1

def process_queue():
    while True:
        number = num_queue.get()
        calculate_sum(number)
        num_queue.task_done()

num_queue = Queue.Queue()
for i in range(thread_count):
    obj_thread = threading.Thread(target=process_queue)
    obj_thread.daemon = True; obj_thread.start()

start_time = time.time()
for int_num in range(10000):
    num_queue.put(int_num)
num_queue.join()

print "thread_count: ", thread_count
print "Start Time (sec): ", start_time
print "End Time (sec): ", time.time()
m,s = divmod(time.time() - start_time,60)
print "Time taken to execute: ", s
```

```
$ python CPU_Bound_MultiThreading.py
thread_count: 1
Start Time (sec): 1511555385.09
End Time (sec): 1511555391.29
Time taken to execute: 6.19999980927
```

```
$ python CPU_Bound_MultiThreading.py
thread_count: 5
Start Time (sec): 1511555408.36
End Time (sec): 1511555422.75
Time taken to execute: 14.388999939
```

CPU Bound Task - Multiprocessing

```
def calculate_sum(num):
    sum_total = 0
    init = 2
    while init <= num:
        sum_total += init
        init += 1

for rand_num in range(10000):
    num_list.append(rand_num)

if __name__ == '__main__':
    start_time = time.time()
    p = multiprocessing.Pool(pool_count)
    p.map(calculate_sum, num_list)

    print "pool_count: ", pool_count
    print "Start Time (sec): ", start_time
    print "End Time (sec): ", time.time()

    m, s = divmod(time.time() - start_time, 60)
    print "Time taken to execute: ", s
```

```
$ python CPU_Bound_MultiProcessing.py
pool_count: 1
Start Time (sec): 1511555710.18
End Time (sec): 1511555716.05
Time taken to execute: 5.87400007248
```

```
$ python CPU_Bound_MultiProcessing.py
pool_count: 5
Start Time (sec): 1511555730.33
End Time (sec): 1511555733.56
Time taken to execute: 3.22299981117
```

- Multiprocessing uses multiple cores (cpu) to execute same process on each core concurrently.
- Have there own memory.
- Processes execute on independent CPU's and maintains there own GIL, hence even the CPU bound task does not have any effect on performance.

The Common Mistake & How to avoid ?



- Need performance improvement?
- **Solution** : Implement Multithreading



- Analyze – The problem statement
- **Solution** – Check ,If multithreading is the answer to it (there can be other reasons for performance latency)
- **Implement**



