Multithreading and Multiprocessing in Python

Petr Zemek

November 15, 2024

Python Guild @ Gen™

Disclaimers

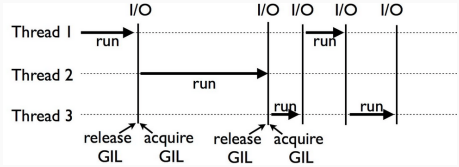
- The talk is accompanied by a lot of examples:

  https://github.com/s3rvac/talks/tree/master/2024-11-15-Multithreading-and-Multiprocessing-in-Python/examples

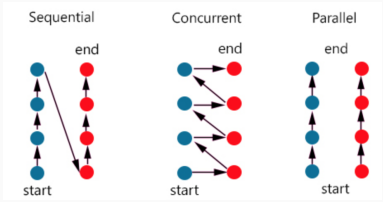
- The talk does not cover asynchronous processing (asyncio), coroutines, green threads, fibers, etc.

General Terms

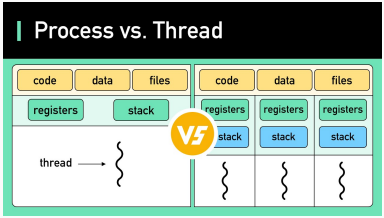
- Concurrency vs parallelism
- Program vs process vs thread
- Global Interpreter Lock (GIL) vs free threading
- Cooperative vs preemptive multitasking
- Thread safety and process safety



(source)



(source)



(source)

Common Misconceptions and Myths

- *"Python does not have threads"* or *"I cannot use threads in Python"*
- *"Python is single-threaded"* or *"Python is not multi-threaded"*
- *"asyncio solves the GIL issues"*
- *"Thanks to GIL, I do not need to use locks"*
- *"I cannot use threads in Python for CPU-heavy computations"*
- *"GIL is part of the language"*
- *"GIL is a Python-specific thing"*
- *"CPython had GIL from the beginning and was never removed"*
- *"Python sucks because of GIL"*

Global Interpreter Lock (GIL)

- Origin
- Benefits
- Drawbacks
- Who has seen it? Let's look at it ;-)
- When does GIL get released?
- Experimental free-threading support in Python 3.13

(source)

How to Create, Run, and Terminate Threads/Processes?

- Option 1: Manual
  - `threading.Thread` and `multiprocessing.Process`
  - `start()`, `join()`, and `terminate()` (only for processes)
  - Possible to inherit `threading.Thread` and `multiprocessing.Process`
- Option 2: Process and thread pools via `multiprocessing`
  - `multiprocessing.pool.Pool` (process pool)
  - `multiprocessing.pool.ThreadPool`
- Option 3: Process and thread pools via `concurrent.futures`
  - `concurrent.futures.ProcessPoolExecutor`
  - `concurrent.futures.ThreadPoolExecutor`
- Other options for processes:
  - `subprocess` module
  - Low-level: `os.system()`, `os.fork()`, and `os.spawn*()`

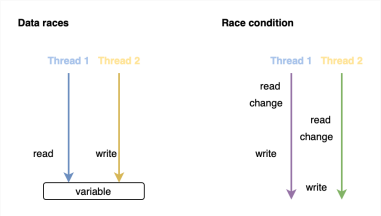
How to Communicate or Share Data Between Threads/Processes?

- Threads:
  - Shared memory (potentially with synchronization)
  - `queue.Queue`, `collections.Deque`
- Processes:
  - `multiprocessing.Queue`
  - `multiprocessing.Pipe`
  - Shared memory: `multiprocessing.{Value, Array}`
  - Server process: `multiprocessing.Manager`
- Generic (both threads and processes):
  - Files
  - Sockets (`socket`)
  - Databases / Redis / cloud buckets / ...
  - RabbitMQ / Kafka / ZeroMQ / ...
  - ...

Common Issues When Using Threads or Processes

Would require a separate talk...

- Data races / race conditions
- Blocking
- Deadlock
- Livelock
- Starvation



(source)

Example: Two Locks

```
// Thread A                  // Thread B
lock1.acquire();             lock2.acquire();
lock2.acquire();             lock1.acquire();
lock2.release();             lock1.release();
lock1.release();             lock2.release();
```

(source)

How to Synchronize Threads/Processes?

Note: Threads and processes have the same synchronization primitives available (`threading` vs `multiprocessing`)

- Lock (also called a *mutex*)
- RLock (*reentrant/recursive lock*)
- Semaphore
- Condition (also called a *condition variable*)
- Event
- Barrier

Note: Do not forget to utilize the `with` statement!

Threads vs Processes

	Threads	**Processes**
Primary use case	Concurrent work (I/O-bound)	Parallel work (CPU-b.)
Shared memory	Yes (direct access)	No (only indirectly)
Create/destroy time	Less expensive	More expensive
Context switch time	Less expensive	More expensive
Lifetime	Depends on the parent process	Independent
Resilience	Crash in a thread crashes process	Independent
Subject to GIL?	Yes	No
Est. worker count	10s–1000s	~number of cores*

Concluding Notes

- Rule of thumb: Use threads for concurrency, processes for parallelism
- Achieving thread safety by avoiding shared state (e.g. TLS or immutable objects)
- Multi-process code can run on multiple machines (depends on the communication)
- Using *daemon threads* for background tasks
- Process creation: fork vs spawn
- Unhandled exceptions from threads
- `threading.Timer`
- Use cases for threads and processes

References and Further Reading/Watching

- https://docs.python.org/3/library/threading.html
- https://docs.python.org/3/library/multiprocessing.html
- https://docs.python.org/3/library/concurrent.futures.html
- https://docs.python.org/3/library/subprocess.html
- https://docs.python.org/3/library/os.html
- https://docs.python.org/3/library/queue.html
- https://docs.python.org/3/glossary.html
- https://docs.python.org/3/faq/library.html#threads
- https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock
- https://docs.python.org/3/howto/free-threading-python.html
- GIL: https://peps.python.org/pep-0703/ (*Making GIL optional in CPython*)
- GIL: https://youtube.com/playlist?list=PLP05cUdxR3KsS3yWl5LRiko2lRAp1XPUd